UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

APLICACIONES BASADAS EN MAPAS

Autor: **David Rosa Clemente**

Tutor: **Daniel Borrajo Millán**

Leganés, octubre de 2015

Título: Aplicaciones basadas en mapas
Autor: David Rosa Clemente
Director: Daniel Borrajo Millán

EL TRIBUNAL

Presidente: Fernando Fernández

Vocal: Ana Iglesia

Secretario: Susana Fernández

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 21 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Resumen

El proyecto trata de eliminar la dependencia con Google Maps para realizar distintos servicios que la plataforma ofrece. Esto es realizado por medio de una API generada valiéndonos de la plataforma colaborativa y gratuita OpenStreetMaps, así como de otras librerías Javascript y distintos plugins.

Abstract

The project aims to remove dependencies with Google Maps and develop different services that the platform offers. This has been made building a new API based on OpenStreetMaps, a free and colaborative platform, as well as several Javascript libraries and plugins.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

1.1 Introducción

El 8 de febrero de 2005 Google lanzó una versión beta de Google Maps antes de convertirse en parte de Google Local, el 6 de octubre de 2005. Permitía al usuario mover el mapa, incluía mapa de carreteras, imágenes en satélite, encontrar ubicaciones, negocios, marcar localizaciones con un indicador, calcular rutas... Google Maps continuó evolucionando hasta nuestros días ofreciendo avances sorprendentes como Street View, que permite al usuario ver imágenes 360 º a pie de calle en un gran porcentaje de vías de las zonas pobladas del planeta.

Son muchas las mashups [2.2.1. Mashup] que han surgido combinándose con Google Maps debido a la potencia e infinitos usos de esta aplicación, potenciando más aún su popularidad. Ejemplo de ello son entre otras muchas: Panoramio, Flickr, Wikiloc, Wikimapia.
Google Maps se ha convertido en una herramienta importante y casi indispensable tanto para el usuario corriente como para gobiernos a la hora del uso más cotidiano hasta investigación o espionaje, es por ello que los mapas web forman parte de nuestro día a día teniéndolo al alcance en cualquier dispositivo una cantidad de información que no hubiéramos imaginado hace décadas.

Por todo esto, el proyecto desarrollado que vamos a tratar en el siguiente documento, surge de la necesidad de sustituir la dependencia establecida con la plataforma de Google Maps a la hora de trabajar con mapas. Esto es debido a que Google Maps convirtió su API en 2012 en un servicio de pago a partir de un determinado tráfico de llamadas a la misma.

Para ello se ha desarrollado una aplicación valiéndonos de OpenStreetMaps [2.2.4 OpenStreetMaps], plataforma colaborativa internacional de mapas editables que permite la creación y distribución de mapas de manera libre. Por medio de llamadas a su API obtenemos servicios antes realizados con la API de Google Maps.

Para desarrollar todas las funcionalidades requeridas también hemos utilizado la librería Javascript Leaflet [2.2.5 Leaflet], así como algunos de los plugins desarrollados para ella.

1.2 Objetivos

El principal objetivo de este proyecto es desarrollar un programa que sea capaz de recibir una entrada por parámetros que le permita ejecutar distintas funcionalidades.

Estas funcionalidades serán servicios que realizaremos valiéndonos de los mapas y API de OpenStreetMaps, así como de otras APIs, librerías Javascript o plugins que nos permitirán replicar estos servicios de forma libre y gratuita.

Las funcionalidades desempeñadas en esta aplicación a través de los medios anteriormente comentados son los siguientes:

- Localización (vía navegador)
- Enrutamiento (vía navegador)
- Mostrar coordenadas de un punto
- Establecer marcas en un mapa (vía navegador)
- Exportar mapa en formato XML
- Exportar ruta en formato GPX

1.3 Estructura de la memoria

Este apartado está destinado a explicar brevemente los puntos desarrollados en los capítulos de este documento.

Capitulo 1:
Se pretende dar una visión global del proyecto realizado, los objetivos establecidos, y los acrónimos usados en el resto del documento para una mayor comprensión.

Capitulo 2:
En el Estado de la Cuestión se tratarán temas relacionados con los sistemas de información geográfica y los servidores de mapas web para hacer posible el desarrollo del proyecto.

Capitulo 3:
Este capítulo muestra las herramientas y técnicas desarrolladas en las fases de análisis y de diseño del proyecto. Además, se mostrarán los aspectos más destacados llevados a cabo en la fase de implementación.

Capitulo 4:
En Planificación y Presupuesto se analizará el Diagrama de Gantt, método utilizado para la planificación, y se desglosarán los costes que han hecho posible el desarrollo del proyecto.

Capitulo 5:
En el capítulo de conclusiones se expondrá el punto de vista subjetivo sobre la realización del proyecto y los problemas encontrados.

Capitulo 6:
Este capítulo muestra la explicación de una serie de términos utilizados a lo largo de la memoria para facilitar su comprensión.

Anexo I:
Se muestran las publicaciones y fuentes consultadas para la documentación y elaboración del proyecto.

Anexo II:

Este capítulo contiene manuales de instalación y usuario para conseguir un mejor entendimiento del funcionamiento del programa desarrollado, aprendiendo a utilizar todas sus funcionalidades.

Capítulo 2

Estado de la cuestión

2.1 Estudio previo

2.1.1 Sistemas de Información Geográfica

Un sistema de información geográfica (también conocido con los acrónimos SIG en español o GIS en inglés) es un sistema de información capaz de almacenar información geográfica para su posterior uso. Estos sistemas nos permiten manipular dicha información con el fin de que el usuario pueda analizar la información espacial, editar datos y gestionarlos siendo capaz de mostrarlos en un mapa.

Esta tecnología es utilizada por un sinfín de sectores para lograr una ventaja estratégica o de planificación. Es utilizada en arqueología, planificación urbanística, marketing, investigaciones científicas, etc.

2.1.1.1 Funcionamiento de un SIG

El funcionamiento de un SIG se puede definir como una gran base de datos que almacena información geográfica. La información es devuelta en dos sentidos, podemos preguntar por un objeto o localización de la que se nos devolverán sus atributos almacenados o se puede preguntar por un determinado registro de la base de datos obteniendo su localización cartográfica.

La principal ventaja que nos reporta la utilización de un SIG es la fácil gestión de la información separada en capas con distintas temáticas, de las cuales podemos disponer de forma independiente. Esto permite una mayor extrapolación de la información para un análisis más sencillo y rápido.

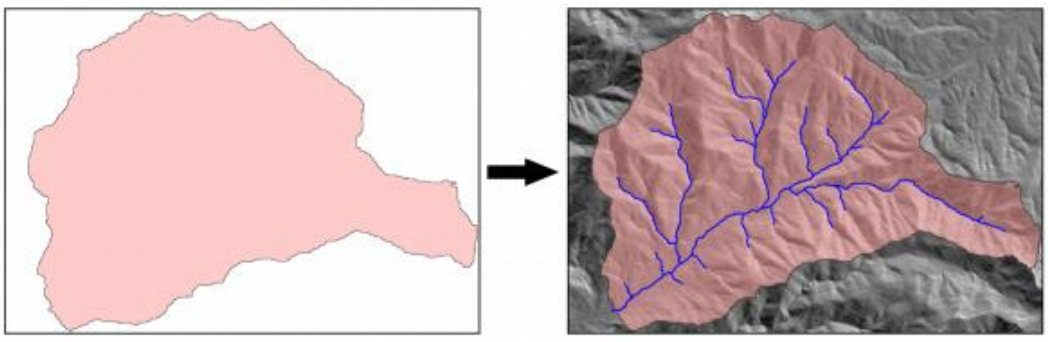


Figura 1. Un SIG puede mostrar la información en capas temáticas [1]

Todo esto, junto con la revolución provocada por las nuevas tecnologías y sus múltiples usos para todo sector que tenga alguna actividad relacionada con un componente espacial, ha provocado una evolución muy grande en los sistemas SIG.

2.1.1.2 Técnicas utilizadas en los sistemas de información geográfica

La información almacenada en una tecnología SIG es digital. La principal fuente de datos llega de la digitalización de imágenes recogidas por satélites o aviones. Estas imágenes son procesadas posteriormente para corregir sus proyecciones y poder ser usadas como datos geográficos.

Los SIG utilizan datos de dos formas: Vectorial o Raster

- **Raster**

Los datos en Raster se caracterizan por representar cualquier tipo de imagen digital por medio de un espacio de celdas regulares dispuestas en filas y columnas donde cada celda solo puede tomar un valor. Esta forma de representación es empleada para la representación de variables continuas en el espacio.

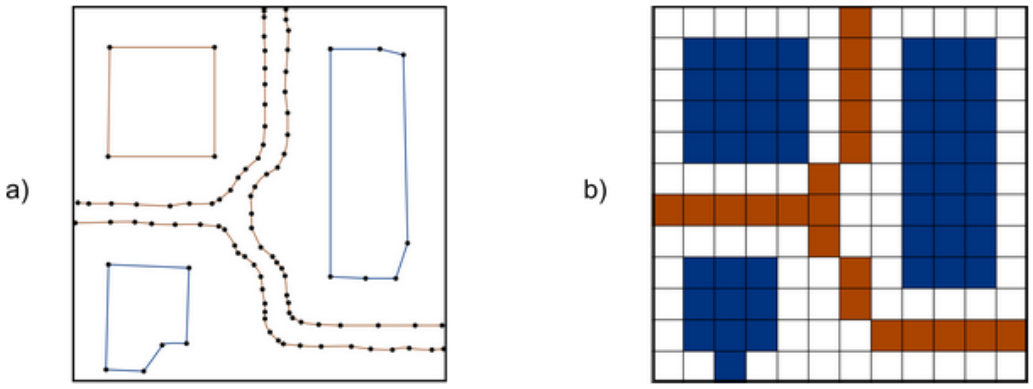


Figura 2. Interpretación vectorial (izquierda) y raster (derecha). [2]

- **Vectorial**

Los SIG vectoriales son los más populares del mercado. Estos expresan las características geográficas por medio de vectores manteniendo las geometrías de las figuras representadas.

Para la representación digital de las entidades del mundo real se utilizan tres elementos: el punto, la línea y el polígono.

- **Puntos**

  Son utilizados para representar una ubicación. También para representar grandes zonas, pero mostradas a una escala pequeña. Como podría ser la ciudad de Madrid en el mapamundi.

- **Líneas**

  Son usadas para elementos lineales como ríos, caminos, líneas topográficas o curvas de nivel. Al igual que en los puntos, en pequeñas escalas pueden representar polígonos.

- **Polígonos**

  Los polígonos representan áreas determinadas de la superficie terrestre. Estos elementos puedes ser parques naturales, lagos, edificios, provincias, etc. Los polígonos contienen la mayor cantidad de información en datos vectoriales.

Figura 3. Dimensión espacial de los datos en un SIG [3]

- **Geocodificación**

Es el proceso por el cual a puntos del mapa (puntos de interés, direcciones, etc.) se les asignan coordenadas geográficas. Este caso será utilizado en este proyecto cuando el usuario introduzca calles, ciudades o puntos de interés y debamos conocer la coordenada exacta para trazar una ruta, poner una marca o devolver la localización exacta. En este caso utilizamos los servicios de Nominatim [3.3.1.1. Nominatim].

Por otra parte, disponemos también de la geocodificación inversa. Como su propio nombre indica, en este caso, devolverá un punto de interés como respuesta a unas coordenadas.

2.1.1.3 Software SIG

Aunque existe software SIG muy especializado con un completo conjunto de aplicaciones, principalmente destinado para gobiernos y grandes empresas, cada vez son más populares las herramientas gratuitas de acceso público que permiten la consulta de información geográfica como pueden ser Google Earth y otros muchos basados en tecnologías web mapping.

En este proyecto haremos uso de las tecnologías web mapping como salida a las distintas operaciones que podemos realizar. Esta tecnología de clientes web SIG nos permite la visualización de datos por medio de mapas cargados de los servidores SIG a través de Internet. Suelen ser clientes ligeros accedidos por medio de navegadores web.

2.1.1.4 El futuro de los SIG

La gran expansión de los sistemas SIG y su integración y uso cada vez mayor en el día a día ha hecho de esta tecnología algo cotidiano y necesario. Además de ser utilizado por universidades, empresas, instituciones, gobiernos que lo aplican para un gran número de sectores, etc., está empezando a proliferar la implantación en Servicios Basados en la Localización (LBS). Estos sistemas hacen uso del GPS integrados en los dispositivos móviles (como pueden ser los teléfonos móviles, tablets, etc.) y que pueden ofrecer puntos de interés o distintos servicios por proximidad a la localización del dispositivo.

2.1.1.5 Cartografía en entornos web

Se ha dado una gran proliferación de entornos web en los últimos años, ofreciendo grandes cantidades de datos cartográficos. Ejemplo de esto son las siguientes plataformas: Google Maps, Bing Maps u OpenStreetMaps.

Es común que estos permitan, a través de su API, crear aplicaciones personalizadas utilizando sus servicios. Estos servicios ofrecen imágenes aéreas, geocodificación, funciones de enrutamiento, etc.

Esta forma de trabajo, vía API, utilizando los servicios de estas plataformas para realizar una aplicación personalizada es la realizada en el proyecto que nos atañe.

El desarrollo de Internet, así como los estándares promulgados por la Open Geospatial Consortium (OGC) [Open Geospatial Consortium 2.1.2] han facilitado e impulsado el uso de las tecnologías web mapping, que trabajan desde servidores de mapas accedidos por medio de un navegador, consiguiendo las características comunes de los SIG tradicionales.

2.1.2 Open Geospatial Consortium

El Open Geospatial Consortium (OGC) nació en 1994, hoy en día está formada por más de 372 organizaciones públicas y privadas. Su propósito es la definición de estándares abiertos e interoperables dentro de los SIG y de la World Wide Web (WWW). Busca facilitar el intercambio de información geográfica en beneficio de usuario.

Las especificaciones más importantes surgidas del OGC son:

- **GML** - Lenguaje de Marcado Geográfico
- **KML** - Keyhole Markup Language es un lenguaje de marcado basado en XML para representar datos geográficos en tres dimensiones.
- **WFS** - Web Feature Service o Servicio de entidades vectoriales que proporciona la información relativa a la entidad almacenada en una capa vectorial.
- **WMS** - Web Map Service o Servicio de mapas en la web que produce mapas en formato imagen a la demanda para ser visualizados por un navegador web.
- **WCS** - Web Coverage Service o Servicio de coberturas en la web (datos raster).
- **CSW** - Web Catalogue Service o Servicio de catálogo.

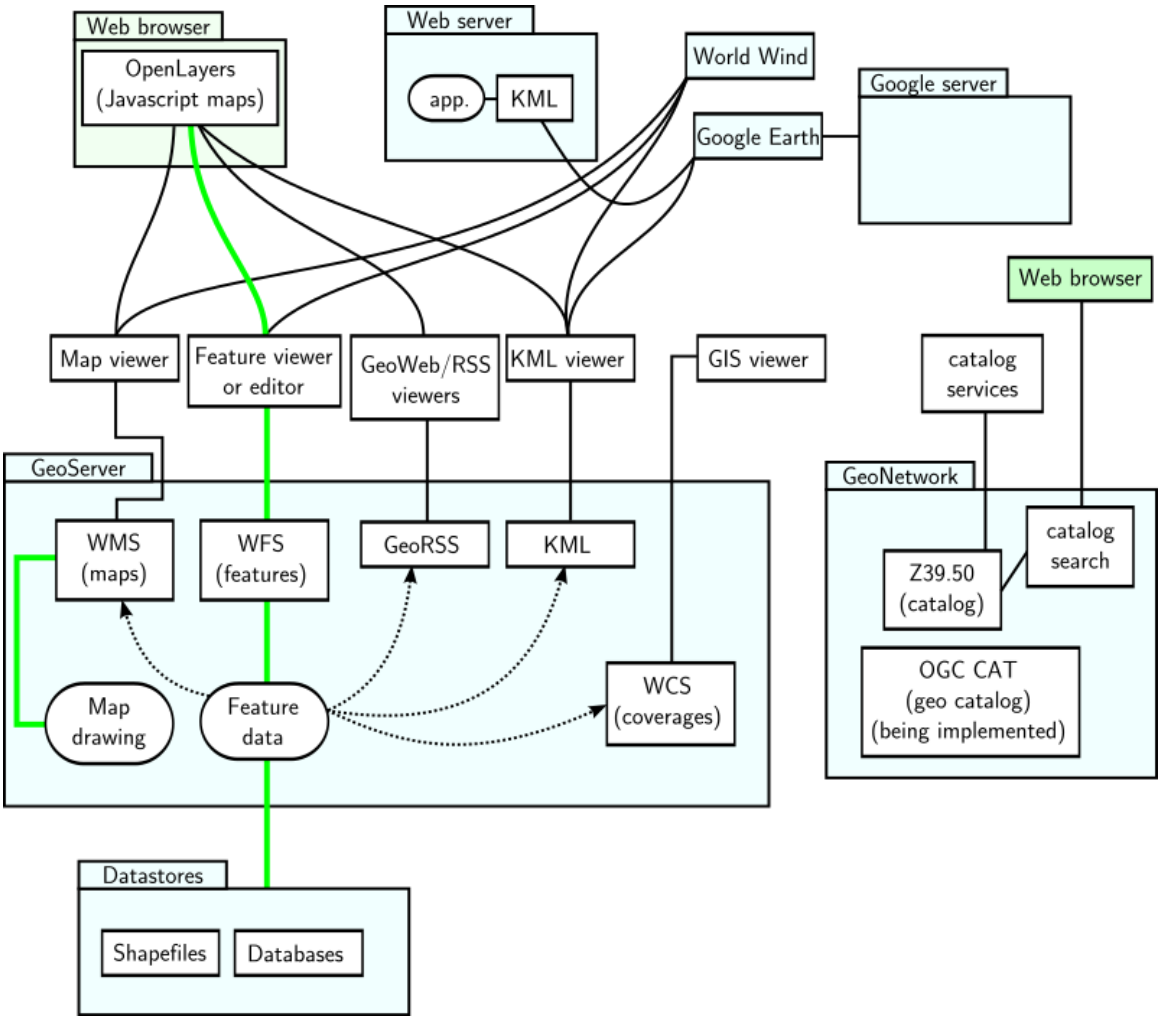


Figura 4. La Fundación fomenta la interoperabilidad entre herramientas geográficas [4]

2.1.3 Servidores de Mapas Web (WMS)

Este estándar es el utilizado en el proyecto a la hora de mostrar las salidas por pantalla utilizando mapas. Es un servicio definido por el OGC que produce mapas de forma dinámica a partir de un conjunto de imágenes en PNG, GIF o JPEG que se apilan por medio de tiles (o cuadrículas) formando el mapa.

El estándar define tres operaciones:

1. Devolver metadatos del nivel de servicio.
2. Devolver un mapa cuyos parámetros geográficos y dimensionales han sido bien definidos.

3. Devolver información de características particulares mostradas en el mapa (opcionales).

Las operaciones WMS pueden ser invocadas por URLs (Uniform Resource Locators) que contengan la información a mostrar en el mapa. Se puede especificar qué porción de mapa mostrar, las coordenadas que centren el mapa, el zoom seleccionado, etc.

Este uso por medio de URLs ha sido el utilizado en este proyecto para, una vez conseguida y tratada la información introducida por el usuario, mostrar el mapa que se solicitaba por entrada de parámetros. Se genera con los datos obtenidos una URL que realiza la petición a la WMS de OpenStreetMaps para el caso de la localización u Open Source Routing Machine (OSRM) en caso de la operación de enrutamiento.

2.2 Tecnologías aplicadas

En este apartado se indican las tecnologías aplicadas en el proyecto

2.2.1 Mashup

Un mashup consiste en la integración de nuevos servicios, de otras aplicaciones web, accedidos vía interface pública o API. Esto permite la integración y reutilización de contenidos y/o funcionalidades de terceros que enriquecen una aplicación.

La arquitectura de un mashup está formada por tres partes:

1. Fuente de datos: es el proveedor de contenidos que pone a disposición la información a través de una API y otros protocolos web como RSS, REST y Web Service.

2. Aplicación consumidora: es la aplicación web que se vale de los datos o funcionalidades de otras para ofrecer un valor añadido.

3. Navegador web cliente: es el interfaz para el usuario por el cual visualiza los contenidos del mashup.

2.2.2 Java

Se ha elegido Java como lenguaje de programación para el desarrollo de la aplicación por la cantidad de herramientas que ofrece para el desarrollo de aplicaciones web.

Además, cuenta con una gran cantidad de librerías gratuitas que sirven de apoyo para realizar el proyecto.

Otra de las características de Java es la posibilidad de ejecutar el código en cualquier tipo de máquina al funcionar sobre una máquina virtual denominada Java Virtual Machine (JVM).

Figura 5. Logotipo de Java

Esta máquina virtual es la encargada de interpretar Java, y ejecutar los ficheros con extensión .class creados por el compilador de Java (javac.exe).

Existen tres ediciones de Java de plataformas diferentes: J2SE, J2EE y J2ME.

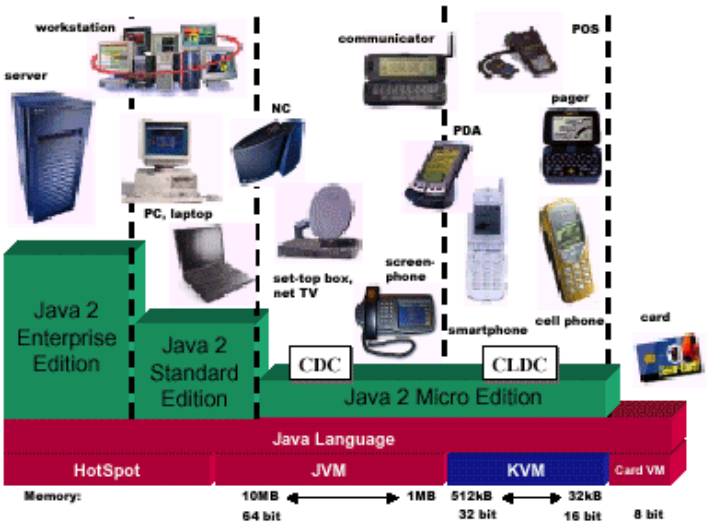


Figura 6. Ediciones de Java [5]

2.2.2.1 Java 2 Platform, Estándar Edition (J2SE)

Esta edición de Java es la más utilizada, y es la que recoge la versión más básica del lenguaje Java.

Contiene un conjunto de ejecución y de API's para crear aplicaciones no destinadas a ser centralizadas y de accesos concurrentes.

2.2.2.2 Java 2 Platform, Enterprise Edition (J2EE)

Orientado a empresas y la integración de sistemas, está pensado para ejecutarse sobre una red de ordenadores de manera distribuida y remota.

La base de J2EE es J2SE.

2.2.2.3 Java 2 Platform, Micro Edition (J2ME)

Versión orientada a pequeños dispositivos móviles (teléfonos, tablets, etc).

2.2.2.4 Conclusiones

Para el desarrollo de este proyecto, como es evidente, se ha utilizado Java en su edición Enterprise Edition, ya que es la que mejor se adaptaba a las necesidades de un proyecto Web, en el que se necesita compartir información a través de la red y además se puede utilizar en cualquier máquina que tenga instalada una máquina virtual de Java.

2.2.3 Eclipse

2.2.3.1 Introducción

Eclipse es el entorno de desarrollo (IDE) utilizado para el desarrollo del proyecto.

Se trata de un software de código abierto que dispone de una amplia comunidad de usuarios que constantemente colaboran para ampliar las funcionalidades de la herramienta.

Figura 7. Logotipo de Eclipse

Con esta herramienta seremos capaces de escribir código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Además, existen una gran cantidad de plugins o extensiones para ampliar funcionalidades.

2.2.3.2 Conclusiones

Para la realización de este Proyecto Fin de Carrera se ha optado por usar Eclipse, ya que contiene ya integrados todos los elementos necesarios para crear una aplicación partiendo desde cero.

Respecto al alcance de este proyecto, al ser tan amplio, se podría decir que abarca todos los aspectos del desarrollo de software, pero realmente se concentra inicialmente en la infraestructura para herramientas utilizadas en la construcción de aplicaciones para entornos de ejecución Web y Java basados en estándares.

2.2.4 OpenStreetMaps

OpenStreetMap (OSM) es un proyecto abierto y colaborativo para crear y editar mapas elaborado por personas de la comunidad.

Es una organización internacional sin ánimo de lucro dedicada a fomentar el crecimiento, desarrollo y distribución de datos geoespaciales libres y a proveer datos geoespaciales a cualquiera para usar y compartir.

Figura 8. Logotipo de OpenStreetMaps

Los mapas se crean usando la información geográfica capturada con dispositivos móviles, ortofotografías y otras fuentes libres. En esta cartografía, tanto las imágenes creadas como los datos vectoriales almacenados en su base de datos, se distribuye bajo licencia abierta Licencia Abierta de Bases de Datos.

2.2.4.1 Motivaciones

Aunque existen en el mercado proveedores de mapas que permiten usar sus servicios de manera gratuita para uso personal, estos imponen restricciones de uso como puede ser el bajo límite de llamadas diarias a la API. OSM aunque no permite llamadas ilimitadas, aumenta considerablemente este límite.

Otra característica que diferencia a OSM de otros proveedores es la posibilidad de edición colaborativa a nivel mundial de los mapas.

Además, integra en una sola base de datos los datos que terceros han liberado uniendo datos de calles, carreteras, edificios,… de todos los países del mundo.

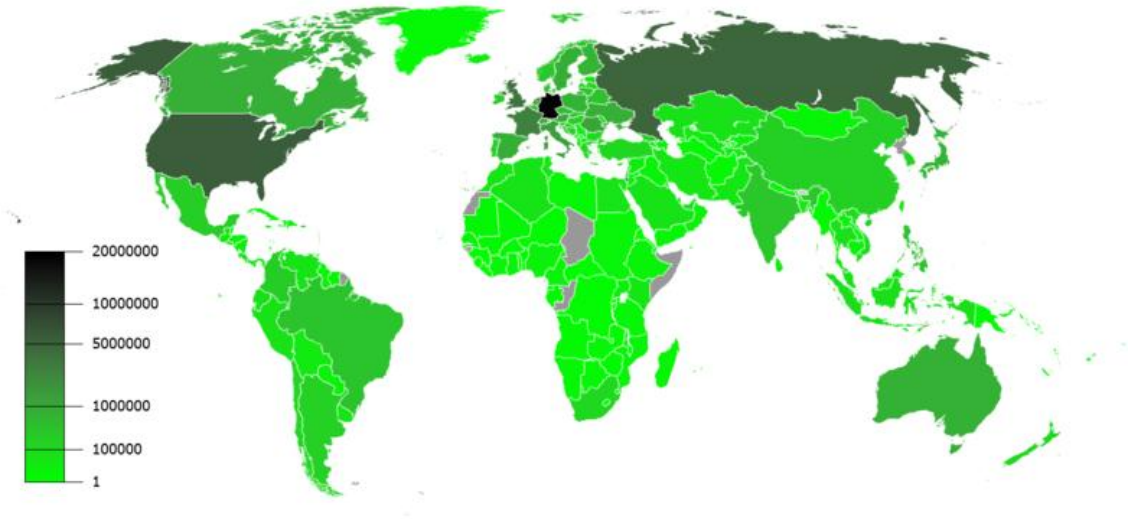


Figura 9. Estadísticas de uso de OpenStreetMap donde se realizan un mayor numero de ediciones por parte de los usuarios [6]

2.2.4.2 Formato de datos

OpenStreetMap utiliza una estructura de datos topológica. Los datos primitivos o elementos básicos de la cartografía de OSM son:

- Los **nodos** (*nodes*). Son puntos que recogen una posición geográfica dada.

- Las **vías** (*ways*). Son una lista ordenada de nodos que representa un polígono.
- Las **relaciones** (*relations*). Son grupos de nodos, vías y otras relaciones a las que se pueden asignar determinadas propiedades comunes.
- Las **etiquetas** (*tags*). Se pueden asignar a nodos, caminos o relaciones y constan de una clave y de un valor.

2.2.4.3 Componentes OSM

OpenStreetMaps dispone de una gran comunidad de desarrolladores que disponen de la API de OSM, la cual está muy completa y bien documentada. Es la propia comunidad la que se encarga de actualizar y documentar la API.

A continuación vamos a ver un diagrama de la relación de los componentes a alto nivel que forman OpenStreetMaps:

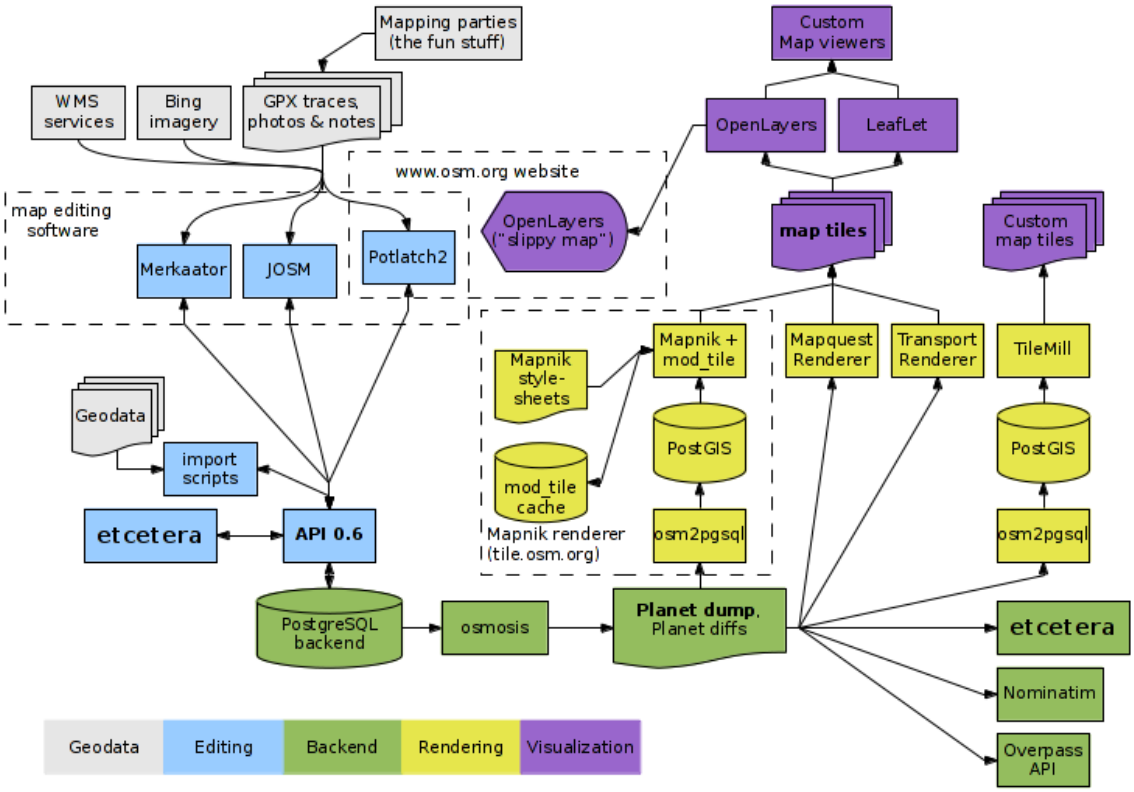


Figura 10. Componentes de OSM [7]

2.2.5 Javascript

JavaScript es un lenguaje interpretado el cual no requiere compilación, utilizado principalmente en el desarrollo y diseño de sitios web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C.

En este proyecto, se usará para incorporar el uso de las librerías de Leaflet [2.2.5 Leaflet] y la correcta generación de los ficheros HTML que contienen los marcadores.

Este lenguaje se puede incluir en cualquier documento y es compatible con cualquier sistema operativo. La mejor manera es incluir JavaScript como un archivo externo, tanto por cuestiones de accesibilidad como por la velocidad en la navegación, este archivo será del tipo *.js y para utilizarlo se debe escribir en el documento HTML:

```
<script type="text/javascript" src="[url del fichero js ]"></script>
```

También es posible incrustar el código en el documento con la etiqueta <script>:

```
<script type="text/javascript"><!--//codigo JavaScript--></script>
```

2.2.6 Leaflet

Es una de las Librerías JavaScript Open Source para crear mapas interactivos en entornos móviles.

Junto con OpenLayers y el API de Google Maps esta es una de las librerías de mapas Javascript más populares y usadas en la mayoría de sitios web tales como FourSquare, Pinterest y Flickr.

Figura 11. Logotipo de Leaflet

En esta ocasión, se ha utilizado esta librería para generar una interfaz web con las herramientas necesarias para poder navegar por los mapas de OSM.

Características

Capas:
- WMS layers
- GeoJSON layers
- Vector layers
- Tile layers
- Marcadores y popups
- Otros tipos de capas son compatibles vía plugins

Interacción:
- Desplazamiento del mapa por inercia al arrastrar
- Zoom con rueda de ratón
- Zoom con doble click
- Zoom a un area (shift + arrastrar)
- Navegación con teclado
- Eventos (click, mouseover,etc)
- Arrastrar marcadores

Visual:
- Animación en zoom y arrastre
- Amigable diseño por defecto
- Compatible con resolución Retina

Personalizable:
- CSS3 popups y controles

- Proyecciones de mapas
- Sencilla interface para personalizar capas de mapas y controles
- Servicios de POO (para extender clases existentes)

Controles:
- Botones de zoom
- Activar y desactivar capas
- Escala

Navegadores compatibles:
- De escritorio:
    - Chrome
    - Firefox
    - Safari 5+
    - Opera 12+
    - IE 7-11
- Móviles:
    - Safari para iOS 7+
    - NavegadorAndroid 2.2+, 3.1+, 4+
    - Chrome
    - Firefox
    - IE10+ para dispositivos Win8

Generales:
- Extremadamente ligera (33 KB de JavaScript)
- Sin dependencias externas

2.2.7 Slippy map

Es el término con el que se conoce a los actuales mapas electrónicos mostrados por pantalla que permite acercar/alejar el zoom y deslizarse por el mapa mediante el ratón realizando un arrastre.

Siendo este tipo de mapa la interface web que permite navegar por la información generada gráficamente devuelta bajo demanda por un servidor de tiles, produciendo un efecto continuo al desplazarse a través del mapa. Los "tiles" son imágenes rectangulares que construyen el mapa.

En este proyecto se ha desarrollado el slippy map con la librería Javascript Leaftlet.

2.2.8 HTML

HTML, HyperText Markup Language (Lenguaje de Marcas de Hipertexto), es el lenguaje utilizado para la construcción de páginas web. Su uso está destinado a describir la estructura y contenido de los sitios web en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Con este lenguaje y a través de etiquetas podemos crear páginas web las cuales podremos editar a través de hojas de estilo y darles funcionalidades con JavaScript.

2.2.9 JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero que surgió para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

El beneficio de JSON, no es que sea más pequeño a la hora de transmitir, sino que representa mejor la estructura de los datos y requiere menos codificación y procesamiento.

2.2.10 XML

XML, sus siglas en inglés significa eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en una forma legible.

Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad, ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

2.2.11 GPX

GPX, o GPS eXchange Format (Formato de Intercambio GPS) es un esquema XML pensado para transferir datos GPS entre aplicaciones. Se puede usar para describir puntos (waypoints), recorridos (tracks), y rutas (routes).

El formato GPX establece un mecanismo estándar para el intercambio y almacenamiento de información de mapas en dispositivos GPS, teléfonos inteligentes y computadoras. Gracias a este formato, el software de un dispositivo puede leer datos creados en diferentes equipos.

Tanto los Routes como los Tracks son colecciones de WayPoints. Un **WayPoint** almacena los datos de una posición por medio de la Longitud y la Latitud.

Las **Routes** están pensadas para describir un camino a recorrer. Son una colección ordenada de puntos que nos permitan describir una ruta a seguir.

Los **Tracks** describen un camino recorrido. Se trata de los puntos que va grabando el GPS cuando lo llevamos activado mientras recorremos el camino. En realidad los Tracks no son una colección de WayPoints, sino una colección de **TrackSegments**, cada uno de los cuales es una colección de WayPoints. Cada Track estará compuesto por uno o más segmentos de track que son los que contendrán los puntos.

Un detalle a tener en cuenta en el esquema formal de los WayPoints es que cuando aparecen como WayPoints aislados en un documento GPX llevan la etiqueta **<wpt>**, mientras que cuando aparecen formando parte de una Route o de un Track llevan las etiquetas **<rtept>** y **<trkpt>** respectivamente.

Capítulo 3

Análisis, diseño e implementación

3.1 Análisis

El proyecto ha pasado por 3 soluciones distintas hasta concretar los requisitos del cliente.

- 1ª SOLUCIÓN: Web
- 2ª SOLUCIÓN: Consola de comandos
- SOLUCIÓN FINAL: Consola de comandos + Web

En un primer momento, se planteó una solución con una interfaz web tanto de entrada como de salida. Al cambiar los requisitos, se decidió desarrollar la aplicación de forma que el usuario solo tuviera la consola de comandos del sistema como interfaz. Finalmente y tomada como solución final, se optó por una mezcla de las dos soluciones anteriores, el usuario seguiría usando la consola de comandos como medio de entrada de datos a la aplicación, pero la salida de datos se hace mediante una página web que muestra los resultados de las búsquedas y además, cuando el usuario lo solicite, se guardaran los resultados en archivos (GPX o XML) en el ordenador desde el que se esté ejecutando la aplicación.

3.1.1 Requisitos del sistema

El formato para describir los requisitos es el siguiente:

IDENTIFICADOR	
DESCRIPCIÓN:	

Los campos utilizados son:

- **Identificador:** RF-XX O RNF-XX donde XX será un número único e identificativo del caso de uso actual y RF se refiere a los requisitos funcionales y RNF a los no funcionales.
- **Descripción**: Descripción de requisito.

Los requisitos que se pactaron con el cliente son:

REQUISITOS NO FUNCIONALES

RNF-01	
DESCRIPCIÓN:	La aplicación debe poder ejecutarse independientemente del Sistema operativo.

Tabla 1. Requisito no funcional - 01

RNF-02	
DESCRIPCIÓN:	La aplicación no requiere configuración.

Tabla 2. Requisito no funcional - 02

RNF-03	
DESCRIPCIÓN:	La aplicación se puede usar sin instalación.

Tabla 3. Requisito no funcional - 03

REQUISITOS FUNCIONALES

RF-01	
DESCRIPCIÓN:	La interfaz de entrada de datos debe ser la consola de comandos del sistema

Tabla 4. Requisito Funcional - 01

RF-02	
DESCRIPCIÓN:	La aplicación debe poder determinar si un lugar o coordenada introducido por el usuario es válido o no.

Tabla 5. Requisito Funcional - 02

RF-03	
DESCRIPCIÓN:	La aplicación deberá poder centrar los resultados en el primer punto que introduzca el usuario.

Tabla 6. Requisito Funcional - 03

RF-04	
DESCRIPCIÓN:	Los mapas o rutas generados de las búsquedas se descargaran automáticamente en la carpeta *output* de la carpeta en la que se encuentra el archivo .jar

Tabla 7. Requisito Funcional - 04

RF-05	
DESCRIPCIÓN:	La aplicación generará un archivo *output.xml* en caso de que se produzca algún error en alguna de las búsquedas que contenga el/los error/es.

Tabla 8. Requisito Funcional - 05

RF-06	
DESCRIPCIÓN:	La aplicación debe devolver un mapa centrado en un lugar o coordenada que introduzca el usuario (**location**).

Tabla 9. Requisito Funcional - 06

RF-07	
DESCRIPCIÓN:	La aplicación debe devolver, si el usuario lo pide, un archivo con los datos del mapa obtenido al introducir un lugar o coordenada (**location_xml**).

Tabla 10. Requisito Funcional - 07

RF-08	
DESCRIPCIÓN:	La aplicación debe pintar en un mapa marcas o puntos (lugares o coordenadas).

Tabla 11. Requisito Funcional – 08

RF-09	
DESCRIPCIÓN:	La aplicación debe pintar en un mapa marcas o puntos (lugares o coordenadas) introducidos por el usuario con sus respectivos iconos y descripciones (**drawmark**).

Tabla 12. Requisito Funcional – 09

RF-10	
DESCRIPCIÓN:	La aplicación deberá generar un archivo con las coordenadas de un lugar introducido por el usuario (**showpoint**).

Tabla 13. Requisito Funcional – 10

RF-11	
DESCRIPCIÓN:	El fichero *output.xml* se usara también para guardar las coordenadas que devuelva la función **showpoint**.

Tabla 14. Requisito Funcional – 11

RF-12	
DESCRIPCIÓN:	Las rutas deben poder hacer a nivel global.

Tabla 15. Requisito Funcional – 12

RF-3	
DESCRIPCIÓN:	La aplicación generará un archivo output.xml que contendrá los datos de tiempo y kilómetros cuando el usuario llame a la función **routing**.

Tabla 16. Requisito Funcional - 13

RF-14	
DESCRIPCIÓN:	La aplicación debe poder trazar rutas entre varios puntos (lugares o coordenadas) introducidas por el usuario y mostrar la ruta sobre un mapa (**routing**).

Tabla 17. Requisito Funcional - 14

RF-15	
DESCRIPCIÓN:	La aplicación debe poder descargar el archivo con la ruta trazada por el usuario si este lo pide (**routing_gpx**).

Tabla 18. Requisito Funcional - 15

3.1.2 Casos de uso

Los casos de uso son una técnica para especificar el comportamiento de un sistema:

"Un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios."

Cada uno de estos casos de uso representa los posibles escenarios en los que usuario y sistema interactúan.

3.1.2.1 Modelo de casos de uso

Figura 12. Leyenda casos de uso

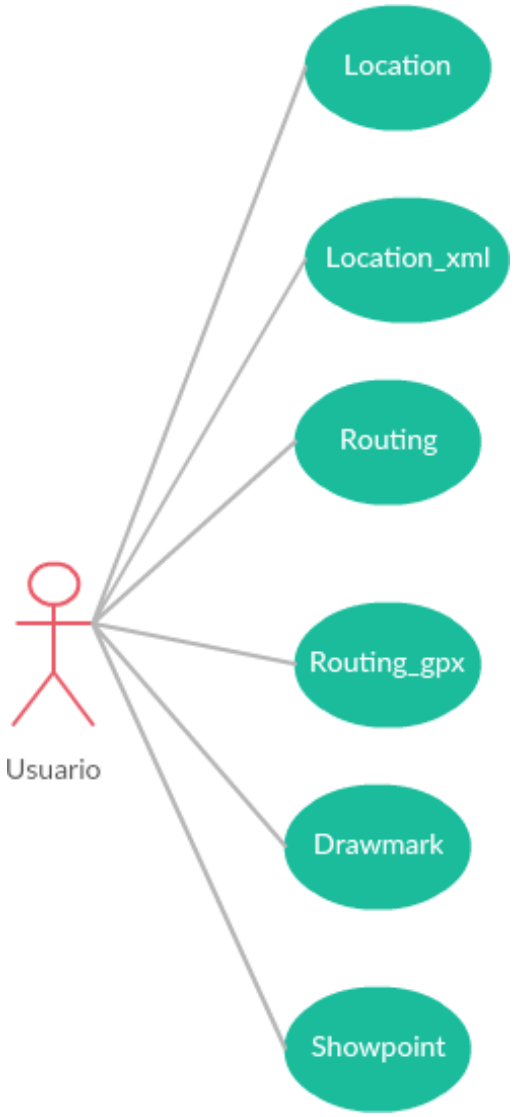


Figura 13. CU: NIVEL 0

3.1.2.2 Descripción de casos de uso

Para cada uno de los casos de uso que se han identificado en el apartado anterior, se va a realizar una descripción más detallada.

El formato para cada caso de uso será el siguiente:

IDENTIFICADOR	
ACTOR	
OBJETIVO	
PRECONDICION	
DESCRIPCION	
POSTCONDICION	

Los campos utilizados son:

- **Identificador:** CU-XX donde XX será un número único e identificativo del caso de uso actual.
- **Actor:** Entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad.
- **Objetivo:** Breve explicación del caso de uso.
- **Precondición:** Condiciones previas que deben cumplirse previamente para que se pueda efectuar el caso de uso
- **Descripción:** Iteración típica entre el actor y el sistema incluyendo alternativas.
- **Postcondicion:** Condiciones que se producen tras la ejecución del caso de uso.

CU-01 LOCATION	
ACTOR	Usuario
OBJETIVO	Obtener un mapa centrada en un punto (lugar o coordenada) introducido por el usuario.
PRECONDICION	• El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. • El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *location*. 3. El usuario introduce ahora el punto que quiere localizar sustituyendo los espacios por '_'. 4. La aplicación comprueba que el punto introducido por parámetros es correcto. 5. La aplicación hace una llamada al api de OSM mediante una URL con las coordenadas que ha introducido el usuario. 5b. El parámetro introducido por el usuario es incorrecto, se avisa al usuario y se genera el fichero *output.xml* con la descripción del error. 6. Se recogen los resultados de la llamada URL, se abre el navegador y se muestra el mapa obtenido.
POSTCONDICION	Ficheros de salida de la carpeta output actualizados.

Tabla 19. Caso de uso - 01

CU-02 LOCATION_XML	
ACTOR	Usuario
OBJETIVO	Obtener un mapa centrado en un punto (lugar o coordenada) introducido por el usuario y además un archivo en formato XML que contenga todos los nodos con la información del mapa mostrado en el navegador.
PRECONDICION	El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *location_xml*.

	3. El usuario introduce ahora el punto que quiere localizar sustituyendo los espacios por '_'. 4. La aplicación comprueba que el punto introducido por parámetros es correcto. 5. La aplicación hace una llamada al api de OSM mediante una URL con las coordenadas que ha introducido el usuario. 5b. El parámetro introducido por el usuario es incorrecto, se avisa al usuario y se genera el fichero *output.xml* con la descripción del error. 6. Se recogen los resultados de la llamada URL, se abre el navegador y se muestra el mapa obtenido. 7. Se crear un fichero *mapa.xml* en la carpeta *output* que contiene la información de todos los nodos reflejados en el mapa.
POSTCONDICION	Ficheros de salida de la carpeta output actualizados.

Tabla 20. Caso de uso - 02

CU-03 ROUTING	
ACTOR	Usuario
OBJETIVO	Obtener un mapa que contenga la ruta entre los puntos introducidos por el usuario.
PRECONDICION	El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *routing*. 3. El usuario introduce ahora al menos dos puntos entre los que quiere realizar la ruta, separados por espacios. 4. La aplicación comprueba que los puntos introducidos son correctos. 4b. Si los puntos no son correctos se genera un fichero output.xml con los puntos erróneos. 5. La aplicación hace una llamada al api de OSM mediante una URL con los puntos que ha introducido el usuario. 5b. Si no se puede generar una ruta entre los puntos, se genera el fichero output.xml con la descripción del

	error. 6. Se recogen los resultados de la llamada URL, se abre el navegador y se muestra el mapa de ruta obtenido. 7. Se genera un fichero output.xml que contiene los datos del tiempo y los kilómetros de la ruta seleccionada.
POSTCONDICION	Ficheros de salida de la carpeta output actualizados.

Tabla 21. Caso de uso - 03

CU-04 ROUTING_GPX	
ACTOR	Usuario
OBJETIVO	Obtener un mapa que contenga la ruta entre los puntos introducidos por el usuario y además un fichero gpx con la información de la ruta.
PRECONDICION	El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *routing_gpx*. 3. El usuario introduce ahora al menos dos puntos entre los que quiere realizar la ruta, separados por espacios. 4. La aplicación comprueba que los puntos introducidos son correctos. 4b. Si los puntos no son correctos se genera un fichero output.xml con los puntos erróneos. 5. La aplicación hace una llamada al api de OSM mediante una URL con los puntos que ha introducido el usuario. 5b. Si no se puede generar una ruta entre los puntos, se genera el fichero output.xml con la descripción del error. 6. Se recogen los resultados de la llamada URL, se abre el navegador y se muestra el mapa de ruta obtenido. 7. Se genera un archivo ruta.gpx con los datos de la ruta introducida.

	8. Se genera un fichero output.xml que contiene los datos del tiempo y los kilómetros de la ruta seleccionada.
POSTCONDICION	Ficheros de salida de la carpeta output actualizados.

Tabla 22. Caso de uso - 04

CU-05 DRAWMARK	
ACTOR	Usuario
OBJETIVO	Obtener un mapa que contenga los marcadores introducidos por el usuario.
PRECONDICION	El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *drawmark*. 3. El usuario introduce ahora al menos un punto que quiere que sea un marcador en el mapa, además del icono y la descripción que quiera en caso de querer customizar cada punto. 4. La aplicación comprueba que los puntos introducidos son correctos. 4b. Si los puntos no son correctos se genera un fichero output.xml con los puntos erróneos. 5. La aplicación busca la ruta de la plantilla HTML de los marcadores (*template.html*) y genera un nuevo fichero *marcadores.html* con el mismo código que la plantilla más los puntos introducidos por el usuario. 6. Se abre el navegador y se muestra el archivo *marcadores.html* que contiene el mapa centrado en el primer marcador introducido por parámetro y el resto de marcadores en caso de haberlos.
POSTCONDICION	• Ficheros de salida de la carpeta output actualizados. • Fichero *marcadores.html* contiene el mapa actualizado con los últimos marcadores.

Tabla 23. Caso de uso - 05

CU-06 SHOWPOINT	
ACTOR	Usuario
OBJETIVO	Obtener las coordenadas de un lugar pasado por

	parámetro.
PRECONDICION	El usuario debe abrir una consola de comandos del sistema desde la raíz de la carpeta en la que se encuentra el .jar. El usuario debe tener conexión a internet y los servidores de OSM han de estar operativos.
DESCRIPCION	1. Se escriben por consola los parámetros generales para ejecutar el .jar: java –jar API.jar API_OSM 2. A continuación escribe el nombre de la función que va a usar, en este caso *showpoint*. 3. El usuario introduce ahora el lugar del que desea obtener las coordenadas sustituyendo los espacios por '_'. 4. La aplicación comprueba que el lugar es correcto y que existe. 4b. Si los puntos no son correctos se genera un fichero output.xml con los puntos erróneos. 5. La aplicación genera un archivo *output.xml* que contiene las coordenadas del punto que ha introducido el usuario.
POSTCONDICION	Ficheros de salida de la carpeta output actualizados.

Tabla 24. Caso de uso - 06

3.1.3 Ciclo de vida

El modelo de ciclo de vida define las fases a través de las cuales pasa un proyecto.

El ciclo de vida del software es una vista de las actividades que ocurren durante el desarrollo del mismo, determina el orden de las tareas o actividades involucradas, así como los criterios de transición entre ellas.

Para este proyecto, el ciclo de vida se basará en el **modelo espiral de Boehm** [8].

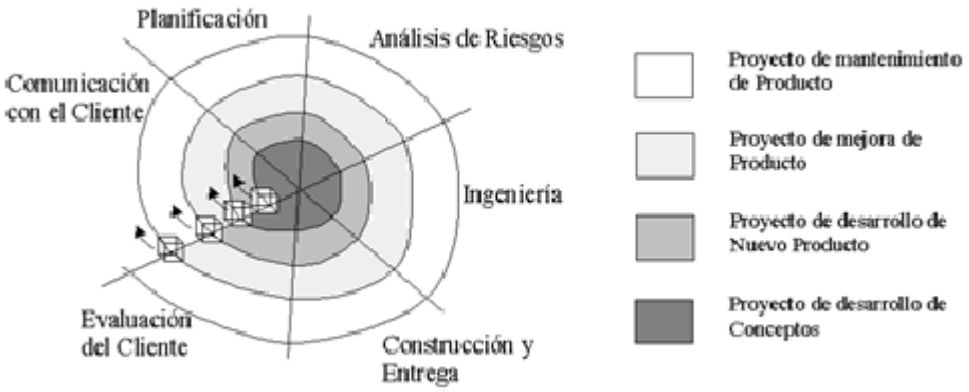


Figura 14. Modelo espiral de Boehm [9]

Este modelo de ciclo de vida es un modelo iterativo que proporciona en cada iteración una versión evolutiva del producto.

En este proyecto se puede distinguir perfectamente cómo tras la fase de evaluación y comunicación del cliente, se ha tenido que volver a realizar una nueva iteración ya que los requisitos iban modificándose hasta llegar a la solución aceptada por el cliente [3.1 ANÁLISIS].

Durante las primeras fases, la versión incremental podría ser un prototipo o un modelo en papel. Durante las últimas iteraciones se producen versiones cada vez más completas del software.

Este modelo se divide generalmente entre tres y seis etapas o regiones. Comunicación con el cliente, Planificación, Análisis de Riesgos, Ingeniería, Construcción y Entrega y Evaluación del Cliente.

Este proceso se inicia desde el centro de la espiral girando en el sentido de las agujas del reloj. Cada ‘circuito’ puede tener varias iteraciones. Sucesivas pasadas

podrían ser usadas para desarrollar un prototipo y progresivamente versiones más sofisticadas del software.

Algunas de las ventajas del modelo en espiral:

- Entender el problema a resolver antes de intentar solucionarlo.
- Analizar las diferentes alternativas y elegir la más adecuada.
- Asegurarse de que el desarrollo del sistema se está haciendo acorde a lo solicitado por el cliente.
- Conocer los riesgos que conlleva el proyecto.

3.2 Diseño

El diseño del proyecto tiene como objetivo definir la arquitectura del sistema y el entorno que le dará soporte. A continuación vamos a ver la arquitectura de la solución y el diseño de la interfaz gráfica.

3.2.1 Arquitectura de sistema

La arquitectura constituye el diseño a más alto nivel de la estructura del sistema.

Para este proyecto, la arquitectura del sistema que se usara será la de Cliente-Servidor. Tenemos una maquina cliente que requiere el servicio de una maquina servidor.

Funcionalmente esta arquitectura permite al usuario final obtener acceso a la información. El cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio).

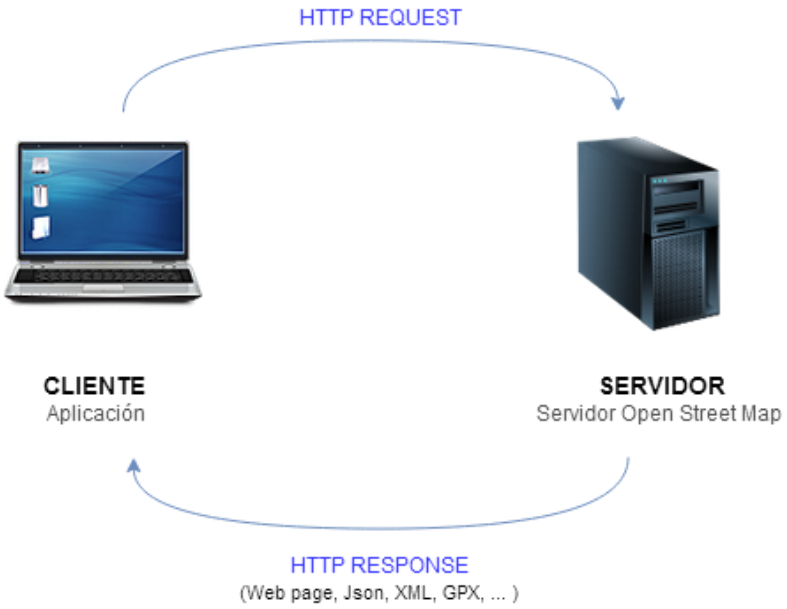


Figura 15. Arquitectura del sistema

3.2.1.1 Características de la arquitectura cliente-servidor

Algunas de las características de esta arquitectura son:

- Cliente y Servidor pueden actuar como una sola entidad y también como entidades separadas, realizando actividades o tareas independientes.
- Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.
- La relación establecida puede ser de muchos a uno, en la que un servidor puede dar servicio a muchos clientes, regulando su acceso a recursos compartidos.
- Los clientes corresponden a procesos activos en cuanto a que son éstos los que hacen peticiones de servicios a los servidores. Estos últimos tienen un carácter pasivo ya que esperan las peticiones de los clientes.
- No existe otra relación entre clientes y servidores que no sea la que se establece a través del intercambio de mensajes entre ambos. El mensaje es el mecanismo para la petición y entrega de solicitudes de servicio.

- El ambiente es heterogéneo. La plataforma de hardware y el sistema operativo del cliente y del servidor no son siempre la misma. Precisamente una de las principales ventajas de esta arquitectura es la posibilidad de conectar clientes y servidores independientemente de sus plataformas.

3.2.2 Interface

En este apartado se va a comentar los distintos cambios de interfaz por los que ha pasado el proyecto tras las distintas reuniones con el cliente y los cambios de requisitos realizados.

Como se comentó en el apartado anterior [3.1 Análisis], el proyecto ha pasado por 3 cambios de interfaz hasta llegar a la solución final. Parte de la interfaz de la solución final se aprovechó de la primera solución. A continuación se explicarán estos cambios.

3.2.2.1 Primera solución

En primera instancia, se diseñó una solución para que toda la interacción del usuario con la aplicación fuera desde una interfaz completamente web.

Figura 16. Interfaz primera solución

[1] Barra de búsqueda. [2] Cuadro de rutas. [3] Cuadro de opciones. [4] Ubicación actual. [5] Zoom +. [6] Zoom -. [7] Minimapa.

Como se observa en la imagen, el usuario disponía de la *[1] Barra de búsqueda* para ejecutar la funcionalidad de **buscar un lugar o coordenada** en el mapa.

Las **rutas** podían manejarse desde el *[2] Cuadro de rutas* donde el usuario introducía un punto de inicio y uno de fin, además, disponía de la opción de añadir más puntos entre medias y cambiar los puntos introducidos de posición. Desde el *[3] Cuadro de opciones* el usuario podía incluir **marcadores** y además dibujar líneas y formas sobre el mapa.
La funcionalidad de **obtener las coordenadas de un lugar** se ejecutaba haciendo doble click sobre cualquier punto del mapa.

Además de las funcionalidades básicas, esta solución incluía un [7] que mostraba una vista reducida del mapa actual, las opciones de [5][6] Zoom y un botón para centrar el mapa en la [4] Ubicación actual en caso de necesitarlo.

La explicación de cómo funciona la aplicación se comentara en el apartado [3.3 Implementación] puesto que esta función web se reutilizo en parte en la solución final.

3.2.2.2 Segunda solución

La segunda solución se diseñó para que tanto los datos de entrada como los de salida se dieran por consola de comandos.

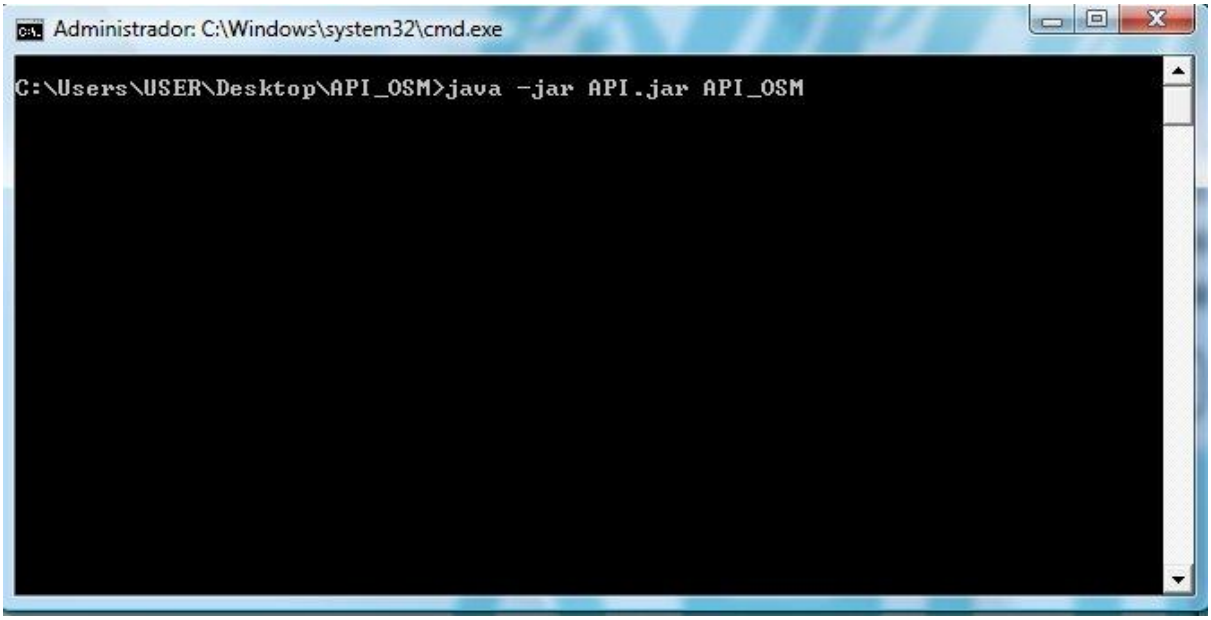


Figura 17. Interfaz consola de comandos segunda solución

Se generó un archivo API.jar para ejecutar el proyecto java en el que se desarrolló esta solución, la cual no proporcionaba una salida de datos visual (web), si no que generaba los archivos XML y GPX de los mapas, rutas y coordenadas y los almacenaba en el ordenador del usuario indicándole además por consola algunos mensajes y/o resultados.

Esta solución se desechó, pues el cliente quería, además de que se descargaran los archivos, obtener un resultado visual de sus consultas.

3.2.2.3 Solución final

Tras la reunión con el cliente en la que se rechazó la solución anterior y se acordaron nuevos requisitos, se planteó la solución final que consistía en una unión de las dos anteriores que podían reutilizarse con algunas modificaciones.

A continuación se muestran algunas capturas de ejecuciones de la aplicación con la salida de datos que se genera de las distintas funciones.

FUNCION LOCALIZAR UN LUGAR O COORDENADA

Sin almacenar el resultado en un archivo de salida

Ejemplo de ejecución:

java –jar API.jar API_OSM location Av._de_la_Universidad,_30,_28911_Leganes,_Madrid

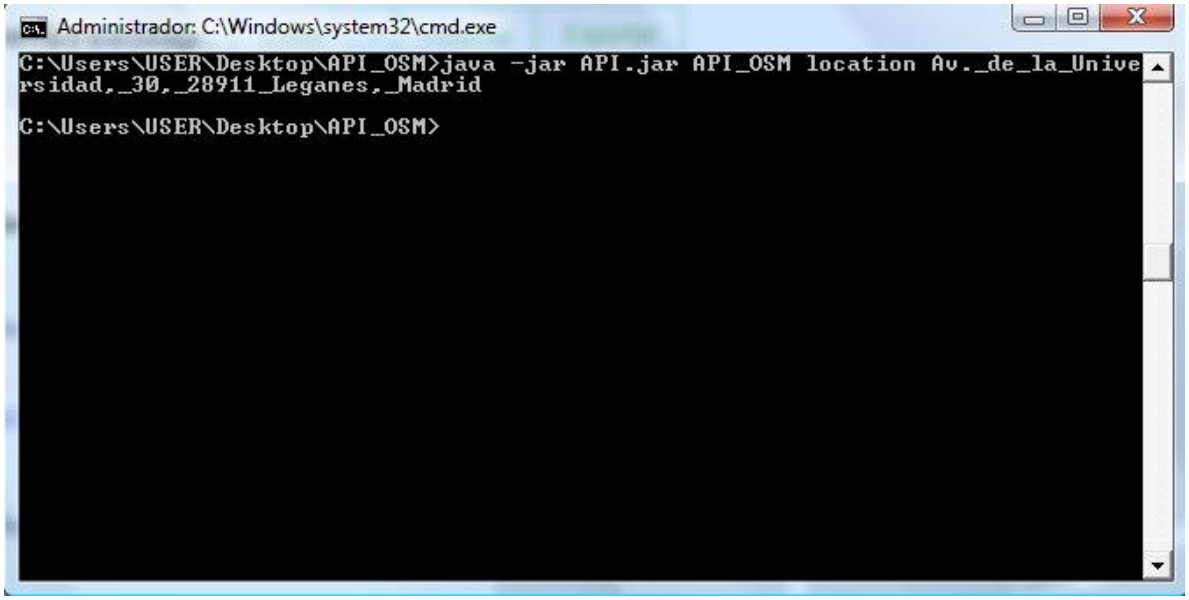


Figura 18. Interfaz consola de comandos ejecución de ejemplo de la funcion localizar un lugar o coordenada.

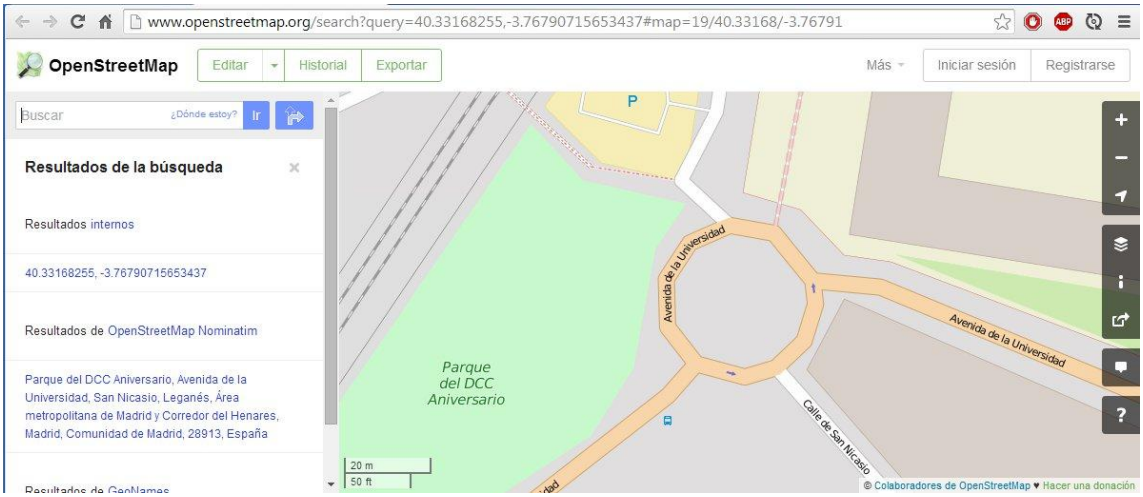


Figura 19. Interfaz resultado ejecución del ejemplo función localizar un lugar o coordenada.

Almacenando el resultado en un archivo de salida

Ejemplo de ejecución:

java –jar API.jar API_OSM location_xml
Av._de_la_Universidad,_30,_28911_Leganes,_Madrid

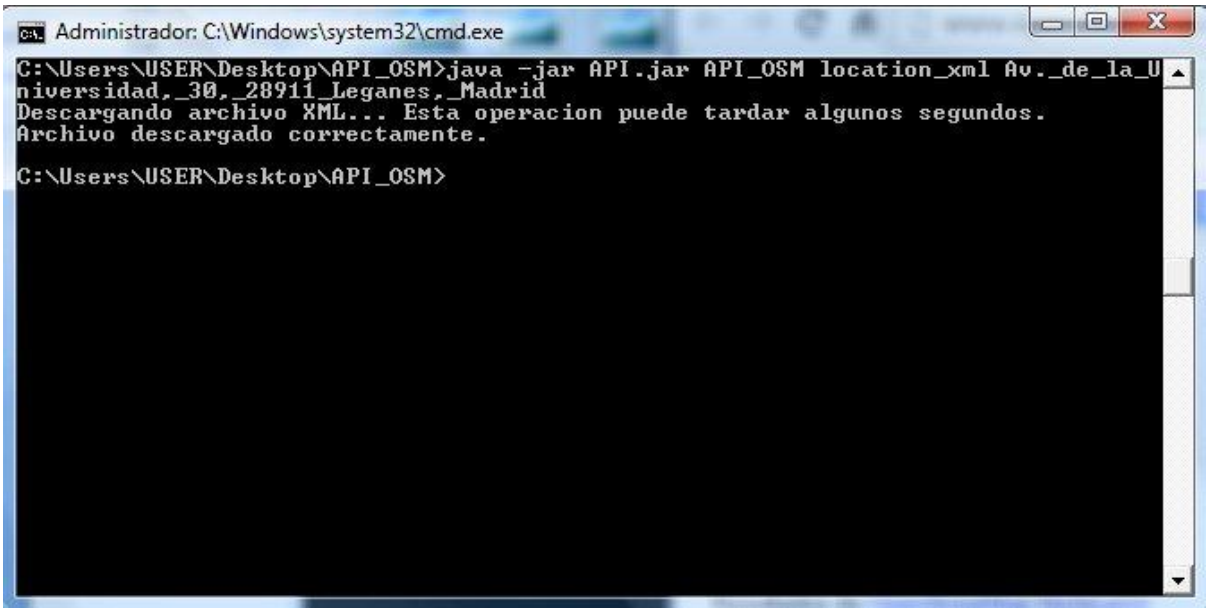


Figura 20. Interfaz consola de comandos ejecución de ejemplo de la funcion localizar un lugar o coordenada almacenando el resultado en un archivo de salida.

La salida web es exactamente igual que en caso anterior, lo único distinto es que se le muestra al usuario un mensaje cuando se está descargando el fichero con los datos y también cuando ha terminado de descargase.

FUNCION TRAZAR RUTAS

Sin almacenar el resultado en un archivo de salida

Ejecución de ejemplo:

java –jar API.jar API_OSM routing Paris,_Francia Versalles,_Francia Amsterdam,_Paises_Bajos Berlin,_Alemania Praga,_Republica_Checa Fussen,_Alemania Venecia,_Italia Florencia,_Italia Roma,_Italia

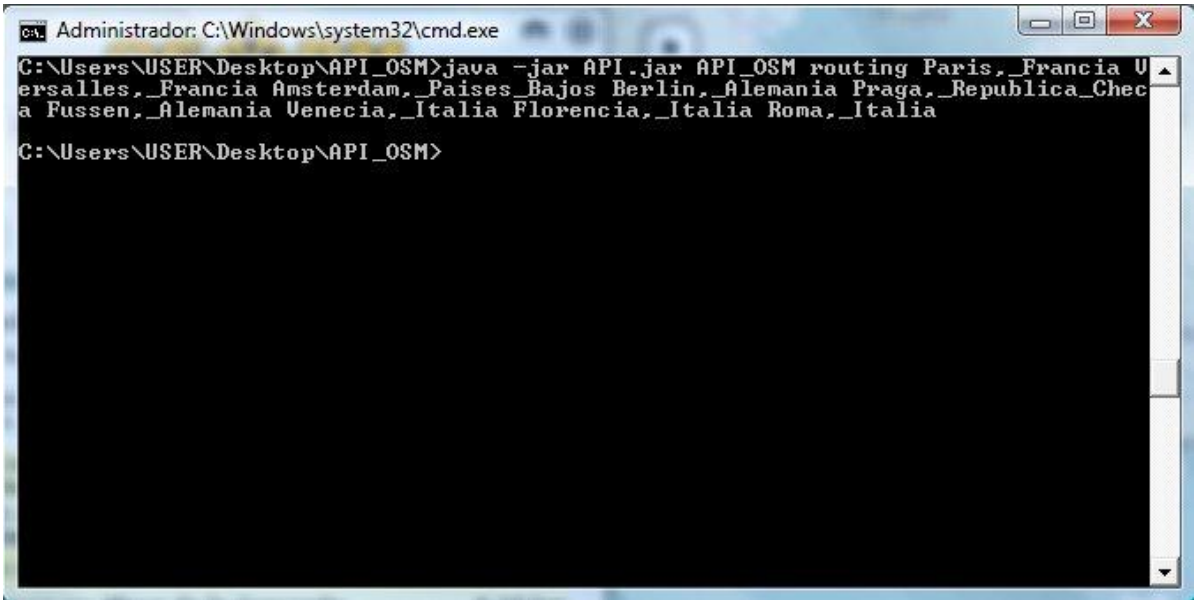


Figura 21. Interfaz consola de comandos ejecución de ejemplo de la funcion trazar rutas.

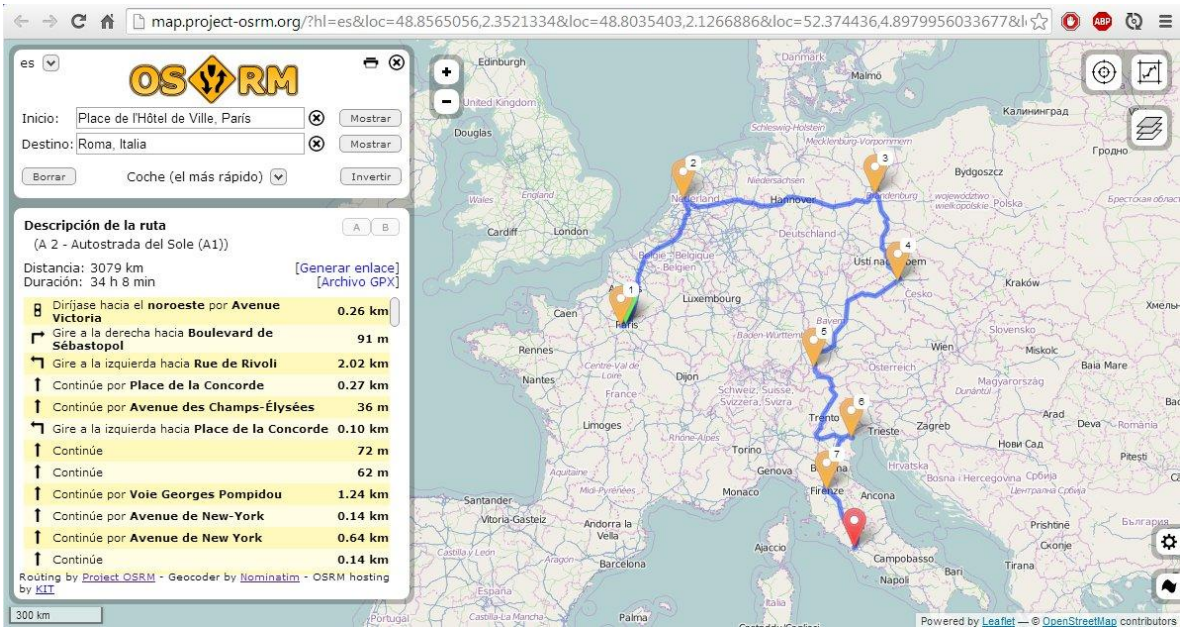


Figura 22. Interfaz resultado ejecución del ejemplo función trazas rutas.

Almacenando el resultado en un archivo de salida

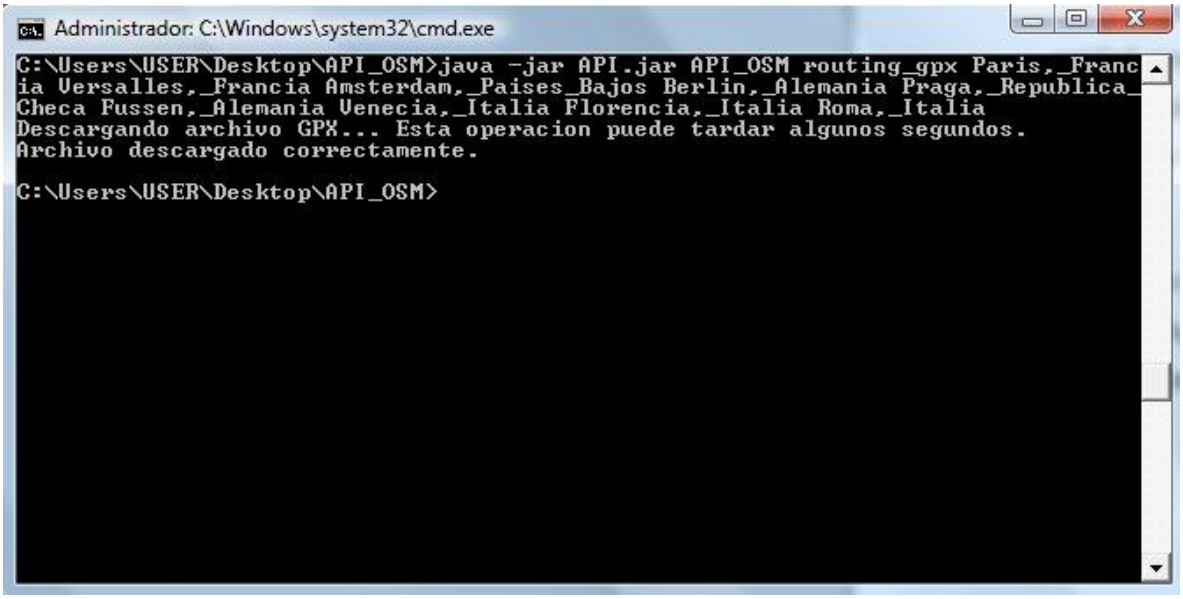


Figura 23. Interfaz consola de comandos ejecución de ejemplo de la funcion trazar rutas almacenando el resultado en un archivo de salida.

La salida web de esta ejecución es la misma que en el ejemplo en el que no se descarga el archivo a diferencia que en la consola se le indica al usuario el mensaje de que se está descargando el archivo gpx que incluye esta opción.

Ambas opciones generan un archivo de salida output.xml con la información de tiempo y distancia de la ruta.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <distancia>3079 km 265 m</distancia>
3  <tiempo>34 h 8 m</tiempo>
4
```

Figura 24. Captura del fichero de salida de la ejecución de ejemplo de la función trazar rutas.

FUNCION CREAR MARCADORES

Ejecución de ejemplo:

java –jar API-jar API_OSM drawmark Trevi_Fountain,_Italia:fuente:Fontana_de_Trevi Coliseo_Roma,_Italia:monumento:Coliseo Ciudad_del_Vaticano:iglesia:Vaticano Castillo_Sant'Angelo,_Italia:monumento:Castillo_de_Sant_Angelo

Figura 25. Interfaz de consola de comandos ejecución de ejemplo de la funcion crear macadores.

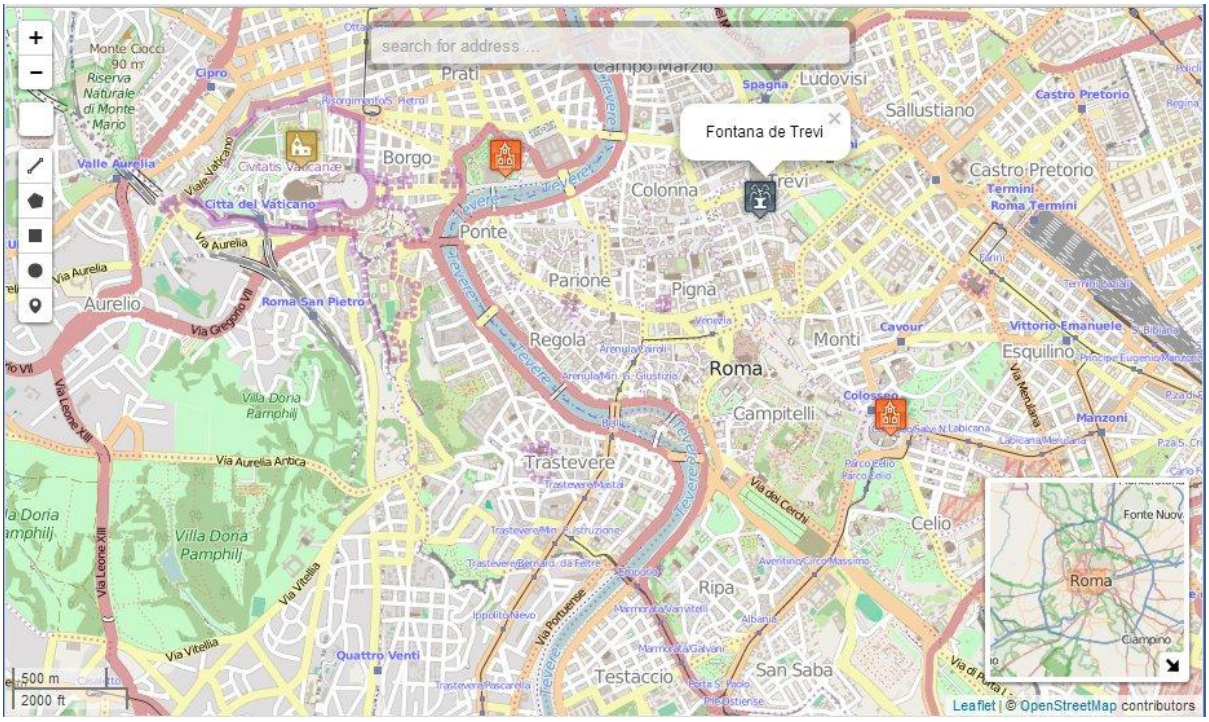


Figura 26. Interfaz resultado ejecución del ejemplo función crear marcadores.

FUNCION MOSTRAR COORDENADAS DE UN LUGAR

Ejemplo de ejecución

java –jar API.jar API_OSM showpoint Av._de_la_Universidad,_Leganés

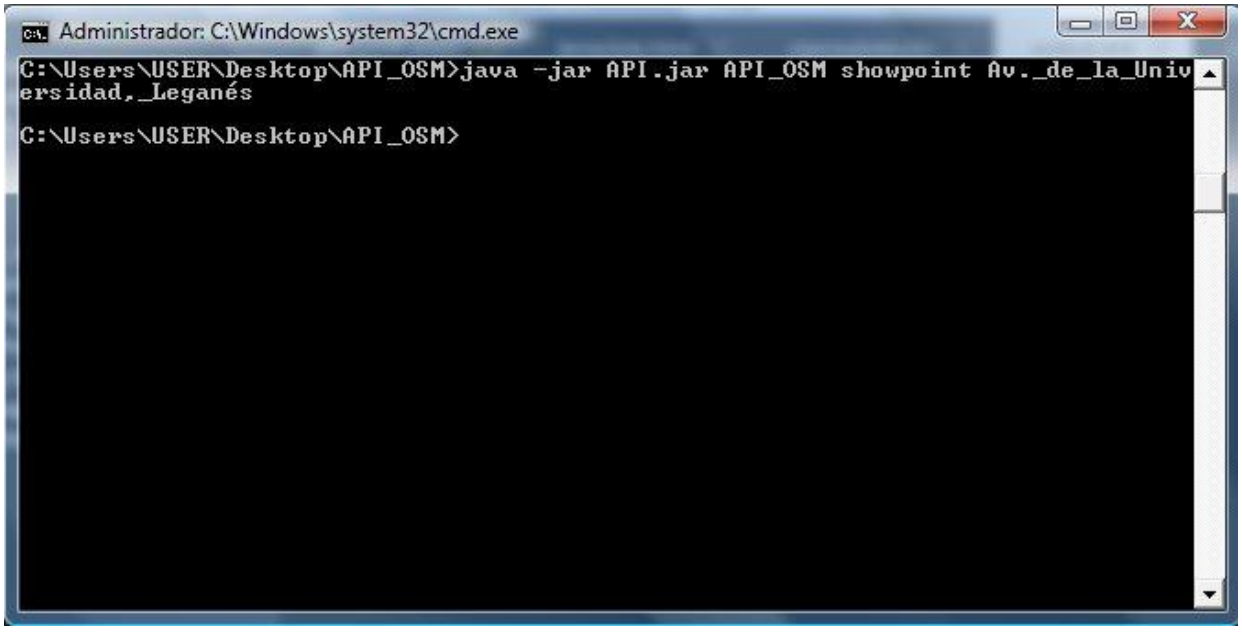


Figura 27. Interfaz de consola de comandos ejecución de ejemplo de la funcion mostrar coordenadas de un lugar.

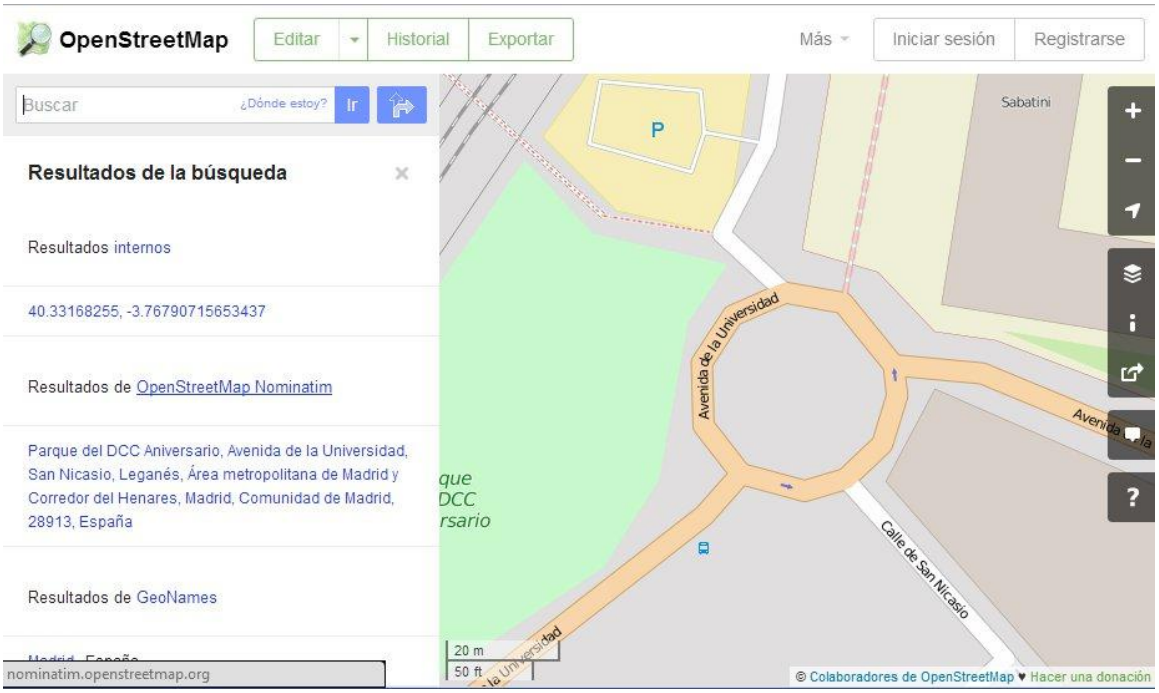


Figura 28. Interfaz esultado ejecución del ejemplo función mostrar coordenadas de un lugar.

.

Además, se genera un archivo output.xml con las coordenadas del sitio

```xml
<?xml version="1.0" encoding="UTF-8"?>
<lugar>Av. de la Universidad, Leganés</lugar>
<coordenadas>40.33168255,-3.76790715653437</coordenadas>
```

Figura 29. Captura del fichero de salida de la ejecución de ejemplo de la función mostrar coordenadas de un lugar.

3.3 Implementación

En esta sección se va a explicar cómo se ha ido desarrollando la aplicación hasta obtener una solución que cumple con los requisitos del cliente.

3.3.1 OpenStreetMap API

El desarrollo de la solución trabaja sobre el api de OpenStreetMap. Por eso hace falta entender los servicios que ofrece y cómo podemos adaptarlos a nuestro proyecto.

3.3.1.1 Nominatim

Es uno de los servicios que más se ha utilizado en el proyecto. Su función principal es buscar datos OSM a partir de un nombre o dirección.

En el proyecto se ha usado para poder saber si el lugar o coordenada que introduce el usuario existe.

Se genera una dirección URL que contiene la búsqueda introducida por el usuario, y además al final de esta cadena se incluye un parámetro que indica que queremos que esta URL nos devuelva un fichero JSON como respuesta

Ejemplo de cadena URL:

http://nominatim.openstreetmap.org/search/Av.%20de%20la%20Universidad%2C%2030%2C%2028911%20Leganes%2C%20Madrid?format=json

Si la búsqueda devolviera un resultado vacío, quiere decir que el lugar o coordenada que ha introducido el usuario no existe. Si por el contrario nos devuelve contenido, podemos seguir trabajando con los datos que ahora tenemos sobre este lugar. Este es el resultado que devuelve la búsqueda anterior:

```
[
  - {
        place_id: "60991061",
        licence: "Data © OpenStreetMap contributors, ODbL 1.0. http://www.openstreetmap.org/copyright",
        osm_type: "way",
        osm_id: "28554896",
      - boundingbox: [
            "40.3315009",
            "40.3318267",
            "-3.7681159",
            "-3.7676957"
        ],
        lat: "40.33168255",
        lon: "-3.76790715653437",
        display_name: "Avenida de la Universidad, San Nicasio, Leganés, Área metropolitana de Madrid y Corredor del Henares,
        Madrid, Comunidad de Madrid, 28913, España",
        class: "highway",
        type: "secondary",
        importance: 0.61
    }
]
```


Figura 30. Captura del fichero JSON que devuelve la llamada anterior al servicio Nominatim.

De todos estos datos, los que nos interesan son la **latitud** y **longitud** de ese punto para poder calcular los límites del mapa en el que vamos a centrar el punto dependiendo también del zoom.

3.3.1.2 OSM Map

Se ha usado este servicio para descargar el contenido del mapa centrado en el punto que el usuario pasa por parámetros.

Utilizando la latitud y longitud obtenidos con el servicio Nominatim, se calcula el bounding box mayor que OSM es capaz de descargar, es decir, el zoom mínimo que soporta dependiendo de la query introducida por el usuario, si esta query contiene a su alrededor muchos nodos, necesitará más zoom que abarque menos nodos.

3.3.1.3 OSRM

Cuando se planteó el problema de las rutas, se tuvo en cuenta que tuviera alcance global y no solo en España, que pudiera exportar los ficheros GPX con las rutas y que pudiera calcular tiempo y distancia de ruta.

Con todos esos requisitos, se eligió OSRM como servicio para calcular las rutas apoyándonos en servicios como Nominatim para comprobar si los puntos que introducía el usuario eran válidos.

3.3.1.4 Leaflet

Al plantear la primera solución puramente web, se hizo uso de la librería Leaflet para interactuar con OSM que funciona con Javascript. Al cambiar de requisitos y necesitar obtener los datos desde la consola de comandos, en principio se desechó esta librería, pero según se fue desarrollando el proyecto, se pensó la opción de volver a utilizarlo para generar los mapas de marcadores.

La idea era tener una plantilla HTML con la estructura básica que contenía las librerías, barra de búsqueda, minimapa,… pero sin marcadores.

Además se incluyeron dos ‘variables’ que posteriormente pudiéramos sustituir desde JAVA. Estas variables son:

- $center, que se sustituirá por el primer marcador valido que haya introducido el usuario en el cual se centrara el mapa.
- $markers, que posteriormente será la lista de marcadores validos del usuario con la información del icono y la descripción.

Estos marcadores se obtenían desde los parámetros que introduce el usuario una vez tratados con algunos de los servicios nombrados anteriormente.

Una vez teníamos la plantilla vacía y la lista de los marcadores, solo faltaba ‘pintarlos’ sobre el HTML. Para ello se copia el código de la plantilla vacía y se sustituyen las variables $center y $markers por sus respectivos valores.

Una vez escrito y guardado el archivo que ahora contiene los marcadores, se guarda con el nombre marcadores.html y queda guardado en la carpeta que contiene el .jar

3.3.2 Pseudocódigo

Antes de empezar a hablar más en detalle del desarrollo de la aplicación en código, se va a añadir el pseudocódigo de las funcionalidades principales de la aplicación.

```
FUNCIÓN main(parámetros)
  SWITCH (operación)
    CASO location:
      SI número de parámetros correcto ENTONCES
        localizacionValida(noExportar);
      SINO
        generarXmlOutput();
      FIN SI
    CASO location_xml:
      SI número de parámetros correcto ENTONCES
        localizacionValida(exportar);
      SINO
        generarXmlOutput();
      FIN SI
    CASO routing:
      SI número de parámetros correcto ENTONCES
        realizarRuta(noExportar);
      SINO
        generarXmlOutput();
      FIN SI
    CASO routing_gpx:
      SI número de parámetros correcto ENTONCES
        realizarRuta(exportar);
      SINO
        generarXmlOutput();
      FIN SI
    CASO drawmark:
      SI número de parámetros correcto ENTONCES
        PARA CADA parámetro en parámetros HACER
          AÑADIR parámetro a listaMarcadores
        FIN PARA

        PARA CADA listaMarcador en listaMarcadores HACER
          DIVIDIR listaMarcadores en trozos separador por
';'
          coord = puntoCorrecto(listaMarcadores[0])
          SI esCoorValidas(coord) ENTONCES
            CREAR Marker
            AÑADIR coord a Marker
            AÑADIR icono a Marker
            AÑADIR descripcion a Marker
            AÑADIR m a marcadores
          SINO
            AÑADIR a marcadoresErroneos
          FIN SI
        FIN PARA

        SI hay marcadoresErroneos ENTONCES
          generarXmlOutput()
        FIN SI

        ABRIR 'template.html'
        LEER 'template.html'
        SUSTITUIR $center por primer marcador

        PARA CADA marcador en marcadores HACER
          AÑADIR marcador a $markers
        FIN PARA

        CREAR FICHERO 'marcadores.html'
          ESCRIBIR en 'marcadores.html' $center y $markers
        CERRAR FICHERO

        ABRIR 'marcadores.html' EN NAVEGADOR
      SINO
        generarXmlOutput()
      FIN SI
    CASO showpoint:
      localizacionValida()
    OMISIÓN:
      generarXmlOutput()
  FIN SWITCH
FIN FUNCION
```

Figura 31. Pseudocódigo función principal

```
FUNCIÓN localizacionValida()
  coordenada = puntoCorrecto()
  SI coordenada es válida
    ABRIR navegador con coordenada

    SI showpoint ENTONCES
      generarXmlOutput()
    FIN SI

    SI exportar ENTONCES
      exportarXML()
    FIN SI
  SINO
    generarXmlOutput()
  FIN SI
FIN FUNCION
```

Figura 32. Pseudocódigo función localizacionValida

```
FUNCIÓN exportarXML()
  Calcular border box para la coordenada
  Lanzar llamada a api que devuelva XML con el mapa
  Descargar XML y guardar en fichero
  DEVOLVER boolean
FIN FUNCIÓN
```

Figura 33. Pseudocódigo función exportarXML

```
FUNCIÓN esCoorValidas()
  Dividir coordenada en latitud y longitud
  Comprobar si los valores están entre -90 - +90 y -180 - +180
  DEVOLVER boolean
FIN FUNCIÓN
```

Figura 34. Pseudocódigo función esCoorValidas

```
FUNCIÓN generarXmlOutput()
  Crear fichero XML
    SI se genera archivo por puntos inaccesibles ENTONCES
        Crear etiqueta <puntoInaccesible> datos del punto </puntoInaccesible>
    FIN SI

    SI se genera archivo por marca erronea ENTONCES
        Crear etiqueta <marcaErronea> datos del punto </marcaErronea>
    FIN SI

    SI se genera archivo por datos de ruta ENTONCES
        Crear etiqueta <distancia> datos distancia </distancia>
        Crear etiqueta <tiempo> datos tiempo </tiempo>
    FIN SI

    SI se genera archivo por error ENTONCES
        Crear etiqueta <mensajeError> datos error </mensajeError>
    FIN SI

    Escribir fichero 'output.xml'
FIN FUNCIÓN
```

Figura 35. Pseudocódigo función generarXmlOutput

```
FUNCIÓN puntoCorrecto()
  SI esCoorValida() ENTONCES
    Sabemos que tenemos una coordenada
  SINO
    Mandamos llamada a Nominatim con el lugar
    SI respuesta es existe el lugar ENTONCES
      Obtenemos las coordenadas del lugar
    SINO
      Devolvemos que no existe el lugar
    FIN SI
  FIN SI
FIN FUNCIÓN
```

Figura 36. Pseudocódigo función puntoCorrecto

```
FUNCIÓN getGPX()
  Construimos una URL para llamar a ROUTER con todos los puntos de la ruta
  Leemos el Json que nos devuelve la llamada

  SI el 'status_message' dice que ha encontrado una ruta entre los puntos ENTONCES
    calcularDatosRuta()
    Devolvemos boolean
  FIN SI

  SI exportarGPX y ruta correcta ENTONCES
    Mandamos una petición a OSRM para que nos devuelva un archivo GPX con la ruta
    Modificamos GPX para usar formato correcto
    Guardamos la ruta en el fichero 'ruta.gpx'
  SINO SI ruta incorrecta ENTONCES
    PARA cada punto en listaPuntos HACER
        Añadimos latitud y longitud del punto a URL
        Comprobamos si ese punto puede hacer ruta con cada uno del resto de puntos
        SI puede hacer ruta con el resto de puntos ENTONCES
          Ese punto es correcto
        SINO
          Ese punto entra en la lista de puntosDudosos
        FIN SI
    FIN PARA
    PARA cada puntoDudoso en puntosDudosos HACER
        Volver a intentar hacer ruta con el resto de puntos
        SI puede hacer ruta con alguno ENTONCES
          Sacar puntoDudoso de la lista
        SINO
          Añadir puntoDudoso a la lista de puntosErroneos
        FIN SI
    FIN PARA
  FIN SI
  Devuelve puntosDudosos
FIN FUNCIÓN
```

Figura 37. Pseudocódigo función getGPX

```
FUNCIÓN calcularDatosRuta()
  Transformar datos de tiempo a H:M:S
  Transformar datos de distancia a Km, m
FIN FUNCIÓN
```

Figura 38. Pseudocódigo función calcularRuta

```
FUNCIÓN realizarRuta()
  PARA cada punto en puntos HACER
    SI el punto es puntoCorrecto() ENTONCES
      Añadir a listaPuntos
    FIN SI
  FIN PARA

  SI no hay puntos que no existan ni coordenadas fuera de rango ENTONCES
    Comprobar si existen puntos dudosos
    SI existen puntos dudosos ENTONCES
      Devolver puntos dudosos
      generarXmlOutput()
    SINO
      Abrir ruta en el navegador
    FIN SI
  FIN SI

  SI un punto no existe o está fuera de rango ENTONCES
    generarXmlOutput()
  FIN SI
FIN FUNCIÓN
```

Figura 39. Pseudocódigo función realizarRuta

3.3.3 Desarrollo de la solución

A continuación se va a dividir en las 4 funcionalidades principales de la aplicación para explicar el desarrollo de cada una de ellas. Además se nombrarán los servicios nombrados en el punto anterior donde han sido previamente explicados.

Las 4 funcionalidades tienen en común la condición de que los datos introducidos por el usuario sean válidos. En caso de no serlos se saca por fichero (output.xml) el error que se ha detectado. Si los datos son correctos, se continúa la ejecución normal la cual se va a explicar en cada apartado.

3.3.3.1 Localizar un lugar o coordenada (location & location_xml)

El objetivo de esta función es obtener las coordenadas de la búsqueda del usuario para poder pasárselo por URL a OSM y que nos muestre en el navegador el mapa centrado en esta búsqueda.

Si el usuario ha introducido una coordenada directamente hay que comprobar que sea correcta. Para ello basta con verificar que la latitud está entre los valores -90.00 y +90.00 y que la longitud está entre -180.00 y 180.00.

Si la búsqueda que ha hecho el usuario es de un lugar literal, haremos uso del servicio Nominatim como se explicó en el punto anterior [3.3.1.1. Nominatim] para obtener las coordenadas de este punto.

Una vez tenemos las coordenadas correctas formamos la URL con la que vamos a obtener el mapa en el navegador que será del tipo:

http://www.openstreetmap.org/search?query=lat,lon#map=zoom/lat/lon

Por defecto se decidió que el zoom que se le pasa al navegador sea 19.

Al usuario se le abre automáticamente el navegador donde visualizar el resultado de su búsqueda.

Hasta aquí la parte común de *location* y *location_xml* y donde acaba la funcionalidad de *location*, pero si el usuario ha decidido además descargar el mapa de su búsqueda (*location_xml*), se produce una llamada adicional al

servicio de mapas de OSM la cual devuelve un archivo .osm del cual se obtienen los nodos del mapa.

La URL que necesitamos formar es del tipo:

http://api.openstreetmap.org/api/0.6/map?bbox=left,bottom,right,top

Donde los valores de left, bottom, right y top se calculan a partir de la latitud y longitud del punto dado. Como se comentó en el punto [3.3.1.2. OSM Map] si la búsqueda del usuario está en un punto con muchos nodos alrededor registrado en OSM y se quiere descargar el mapa con mucho detalle (poco zoom), el servidor de OSM no será capaz de resolver esta petición.

Por tanto, se calcularon unos valores fijos con los que establecer el zoom a tamaño 15,16 y 17 a partir de las coordenadas de la búsqueda y empezando por el menor zoom, 15, se realizarán peticiones al OSM. Si un zoom da error en el resultado, se aumenta el zoom volviendo a calcular los valores left, bottom, right y top y generando una nueva llamada.

En la mayoría de los casos no hace falta pasar de zoom 16 para poder descargar el archivo del mapa correctamente, pero por si acaso se hicieron los cálculos del zoom 17 para tener un margen de error.

Una vez la respuesta que obtenemos es válida, se transcribe esta información en un fichero XML (mapa.xml) el cual se guarda en la carpeta output donde se encuentra el .jar de la aplicación.

3.3.3.2 Trazar rutas (routing & routing_gpx)

Con esta funcionalidad se quiere poder trazar rutas entre al menos dos puntos.

Para ello en un primer lugar se vuelve a hacer uso del servicio Nominatim para comprobar que cada uno de los puntos existe y es válido. Una vez se ha verificado que los puntos son correctos, se envían en una URL formada de la siguiente manera:

http://router.project-osrm.org/viaroute?
loc=lat,lon&loc=lat,lon[&loc=lat,lon]*
&alt=true&instructions=true&output=json

Se pasa un parámetro '&output=json' que nos devuelve el resultado de la ruta en formato JSON en caso de que se haya podido realizar la ruta entre esos puntos. En tal caso se busca el objeto 'route_summary' que contiene los datos de tiempo y distancia total de ruta.

Figura 40. Captura del objeto 'route_summary' del fichero JSON que devuelve la llamada anterior al servicio router

Estos datos se procesan y almacenan en el fichero output.xml con esos datos para que el usuario pueda verlo.

Una vez se ha guardado el fichero, se forma una nueva URL para abrir el resultado de la ruta en el navegador. La URL es del tipo:

http://map.project-osrm.org/?hl=es
&loc=lat,lon&loc=lat,lon[&loc=lat,lon]*
&z=zoom&alt=0&instructions=true&df=0&re=0

El problema que se presenta en las rutas es que puede que todos los puntos sean correctos, validos y existan en OSM, pero no se pueda realizar una ruta entre ellos. Por eso, cuando la primera llamada devuelve como resultado un JSON con el objeto '*status_message'* indicando que no se puede encontrar

una ruta entre los puntos, hay que averiguar cual punto o puntos son los que no pueden unirse con el resto.

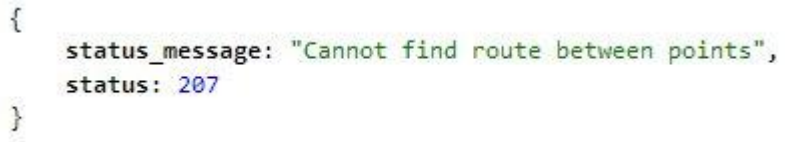


Figura 41. Captura del objeto 'status_message' del fichero JSON que devuelve la llamada anterior al servicio router cuando no puede establecerse una ruta entre los puntos dados.

Una vez se han encontrado los puntos 'erróneos', se guardan en el archivo output.xml para que el usuario pueda verlos.

Si además de visualizar la ruta en el navegador el usuario quisiera descargar el archivo gpx con la información de esta (**routing_gpx**), se forma adicionalmente una URL que devuelve como respuesta dicho archivo, el cual almacenamos en la carpeta de *output* como *ruta.gpx*

La URL que genera el archivo GPX es del tipo:

http://router.project-osrm.org/viaroute?
loc=lat,lon&loc=lat,lon[&loc=lat,lon]*
&alt=true&instructions=true&output=gpx

3.3.3.3 Crear marcadores (drawmark)

El objetivo de esta función es pintar en un mapa los marcadores que introduzca el usuario por parámetro con la posibilidad de personalizar el icono y la descripción de los mismos.

Como se comentó en apartados anteriores, la primera solución que se propuso era puramente web por lo que se tenía un prototipo con todas las funcionalidades incluida la de los marcadores. Por eso se pensó en reutilizar la parte de los marcadores como salida web.

La entrada de datos por consola para esta funcionalidad consiste en una serie de marcadores (lugares o coordenadas) los cuales como se ha comentado, son personalizables de la siguiente manera.

marcador:icono:descripción

Cada uno de los marcadores se toma como una cadena la cual se divide por ‘trozos’ separados por el símbolo dos puntos ‘:’. La parte del marcador se comprueba que sea válida con el servicio Nominatim. Si es correcto se comprueba que el icono que ha elegido el usuario se encuentra en la lista de iconos que se han incorporado a la aplicación (aeropuerto, calle, casa, cine, deportes, fuente, hospital, hotel, iglesia, monumento, pAmarillo, pAzul, pGris, pMorado, pRojo, pVerde, restaurante, ruinas, universidad) y también si ha elegido poner una descripción a ese marcador o por el contrario deja esa opción ‘en blanco’.

Previamente [3.3.1.4. Leaflet] se ha hablado de esta librería la cual se ha usado para pintar vía HTML y JAVASCRIPT los marcadores. A continuación se va a explicar la estructura de la plantilla (*template.html*) que se ha utilizado.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
 <head>
  <title>API OSM</title>
  <!--
    SE INCLUYE EL CONJUNTO DE HOJAS DE ESTILO (CSS)
    Y SCRIPTS (JAVASCRIPT) NECESARIOS PARA EL USO DE
    LOS PLUGINS NECESARIOS DE LEAFLET
   -->
 </head>
 <body>
  <!-- LA CAPA CON ID="Map" CONTIENE EL MAPA QUE SE MUESTRA EN
PANTALLA -->
  <div id="Map" style="position: absolute; top: 0; left: 0; width: 100%; height:
100%;"></div>
  <script>

    $center

    // Añadimos botones cuadro de herramientas
    // Añadimos botones del zoom

     $markers

     // Añadimos el mini mapa
     // Añadimos la barra de búsqueda
    </script>
 </body>
</html>
```

Figura 42. Código simplificado del fichero plantilla template.html.

Todo el código HTML de la plantilla se almacena en una variable String en JAVA.

Las variables **$center** y **$markers** que se incluyen dentro de la etiqueta <body> del HTML guardaran los datos de los marcadores del usuario.

La variable **$center** se sustituye por el primer marcador que será el centro del mapa. El código por el que se sustituye esta variable será del tipo:

```
var map = L.map('Map').setView(new L.LatLng(lat,lon), zoom);
```

Los valores de lat y lon son la latitud y longitud de este primer punto y el zoom por defecto será 13.

Por otra parte, la lista de marcadores, incluido este primero usado para centrar el mapa, serán del tipo:

```
L.marker([lat,long],{
      icon: L.icon({
      iconUrl: 'Ruta Icono Personalizable' ,
      iconSize: [30, 34],
      iconAnchor:   [5, 30],
      popupAnchor:  [14, -30] }),
     title: 'Descripción Marcador'})
.addTo(map)
[.bindPopup('Descripción Marcador')];
```

Los valores que pueden cambiar para cada marcador son:

- **Lat,lon:** Latitud y longitud del punto actual.
- **Ruta Icono Personalizable:** NombreDelIcono.png, si no ha elegido ningún icono, se utiliza uno por defecto.
- **Descripción Marcador:** Descripción del marcador elegido por el usuario, si no ha elegido ninguna descripción esta opción quedará vacía.

Una vez se ha sustituido el valor de todos los marcadores en la variable String en la que almacenó el valor inicial de la plantilla, se crea un archivo (**marcadores.html**) en el que se almacena el conjunto de código ya editado y es el que se mostrará en el navegador.

El usuario puede guardar este archivo *marcadores.html* en otra carpeta y renombrarlo si quiere conservar este mapa de marcadores.

3.3.3.4 Mostrar coordenadas de un lugar (showpoint)

El objetivo de esta función es buscar las coordenadas de un lugar y guardar el resultado en un fichero *output.xml.*

Para ellos se hace uso del servicio Nominatim [3.3.1.1. Nominatim] y se guardan los valores de latitud y longitud que este devuelve junto con el nombre del lugar.

Además, se forma la URL que abre el resultado en el navegador al igual que con la función ***location***

http://www.openstreetmap.org/search?query=lat,lon#map=zoom/lat/lon

Capítulo 4

Planificación y presupuesto

4.1 Planificación

El proyecto comenzó el día 3 de febrero con la primera reunión con el cliente. Tras plantear las primeras bases de lo que se buscaba se realizó una planificación del tiempo que se tardaría en hacer las diferentes tareas de las que consta el desarrollo de un proyecto de este tipo. Se ha fijado una jornada de trabajo estimada en 2 horas de lunes a viernes durante todos los meses del proyecto, salvo Mayo que por motivos de viajes se le dedicaran 3 horas por semana.

Debido a que el proyecto ha tenido varios cambios en la forma de enfocar la solución de los requerimientos a lo largo de su ciclo de vida, se ha sido realizado con la metodología del ciclo de vida en espiral. Ha sido dividido en las fases que explicamos a continuación:

1. Comunicación con el cliente
2. Planificación
3. Análisis de riesgos
4. Ingeniería
5. Construcción y entrega
6. Evaluación por el cliente

Tras efectuar esta serie de fases, en el caso de que tras la evaluación por parte del cliente no fuera totalmente satisfactoria, el ciclo comienza de nuevo para realizar las mejoras del software pertinentes hasta alcanzar un nivel de satisfacción óptimo.

Las pruebas fueron constantes y paralelizadas durante la construcción del proyecto.

A continuación se muestra un diagrama de Gantt con la planificación estimada a principio del proyecto:

Figura 43. Diagrama GANTT

4.2 Presupuesto

En este apartado se va a mostrar de manera detallada el presupuesto desglosado del proyecto, especificando los diferentes gastos que han sido necesarios para su realización.

4.2.1 Gastos de personal

En primer lugar calcularemos el gasto de personal, en el que se analiza el dinero destinado a los recursos empleados en el proyecto. Todos los costes son calculados sin I.V.A

Rol	Nº de horas	Coste (€/hora)	Total (€)
Jefe de Proyecto	86	50	4.300
Analista	106	40	4.240
Desarrollador	160	25	4.000
	TOTAL		**12.540**

Tabla 25. Gastos de personal

Según la planificación del proyecto un recurso trabaja 40 horas mensuales.

4.2.2 Gastos de Hardware

Después valoramos el gasto referente al material amortizable entre los que se encuentran los productos hardware y productos software empleados para el desarrollo del proyecto. Estos precios son sin el I.V.A. incluido.

Descripción	Coste (€)	% Uso Proyecto	Dedicación (meses)	Periodo Depreciación (meses)	Coste Imputable (€)
Ordenador Asus Inter ® Core i5-4210U 1.70GHz 2.40 GHz	575	100	8	60	76,67
Monitor TFT LG L1980 TQ	150	100	8	60	20
Ratón Logitech M185	19'90	100	8	60	2,65
		TOTAL			**99,32**

Tabla 26. Amortización de Hardware

4.2.3 Gastos indirectos

También hay que valorar los gastos indirectos, como luz, agua, fibra óptica, etc. que son estimados como un 5% de los gastos directos. Calculamos los gastos directos del proyecto:

Concepto	Importe (€)
Gastos Personal	12.540
Amortización Hardware	99,32
TOTAL (Gastos directos)	**12.639,32**

Tabla 27. Gastos directos

Una vez conocidos los gastos directos calculamos el 5% de los mismos para obtener los gastos indirectos.

Total gastos indirectos (€)	**631,96 €**

Tabla 28. Gastos indirectos

4.2.4 Riesgo y beneficio

Por último, se va a detallar el coste proporcional al riesgo aplicado al proyecto y el beneficio aplicado al mismo. En este caso, se aplica un 20% de riesgo y un 15% de beneficios. Antes de detallar el coste, se debe indicar el coste total sin el riesgo y los beneficios aplicados:

Concepto	Importe(€)
Gastos Personal	12.540
Amortización Hardware	99,32
Otros gastos	631,96
TOTAL	**13.271,28**

Tabla 29. Gastos proyecto

Aplicamos los porcentajes correspondientes al riesgo y beneficio del proyecto:

Riesgo 20%	Beneficio 15%
2.654,25 €	**1.990,69 €**

Tabla 30. Costes riesgo y beneficio

4.2.5 Presupuesto final

Para concluir con el presupuesto calculamos el presupuesto final del proyecto aplicando el I.V.A a los costes del proyecto. En la siguiente tabla se detalla el coste final:

Presupuesto Final	
Concepto	**Importe (€)**
Gastos proyecto	13.271,28
Riesgo (20%)	2.654,28
Beneficios (15%)	1.990,71
Total (sin I.V.A)	17.916,39
I.V.A (21%)	3.762,44
TOTAL	**21.678,83**

Tabla 31. Presupuesto final

Capítulo 5

Conclusiones

5.1 Conclusiones

Este proyecto se fundamentaba en la realización de un programa que realizara servicios de mapas web. Debido a los diferentes puntos de vista con los que se ha enfocado el proyecto durante su evolución, se han visto varias formas de implementación de los mismos.

Se ha trabajado tanto con un slippy map generado por medio de una librería JavaScript, servicios de web mapping, salidas por ficheros y la combinación de ellos para alcanzar la solución final.

Utilizando herramientas de libre uso, que en el caso de OpenStreetMaps se concibe desde la colaboración de los usuarios y la cesión de datos por medio de múltiples organizaciones para hacerlo posible. Leaflet es una librería OpenSource con muchos plugins implementados de forma colaborativa por usuarios que quieren aportar funcionalidad a la herramienta.

La API de OpenStreetMaps, API de OSRM y la librería Leaflet nos permite implementar un programa utilizando sus servicios sin necesitar mantenimiento de servidores por nuestra parte, dado que solo debemos lanzar las peticiones necesarias.

Este tipo de herramientas colaborativas cada vez son más frecuentes, facilitando el trabajo a la hora de realizar implementación de distintas funcionalidades y dando gran riqueza a la misma con una barrera de aprendizaje pequeña.

5.2 Posibles ampliaciones

Aunque la aplicación no está pensada para que el usuario almacene mapas ni rutas, una posible ampliación y mejora podría ser:

- **Opción de gestión de mapas, rutas y marcadores.**

Esta mejora supondría la incorporación de una base de datos, pero permitiría al usuario poder nombrar sus mapas, modificar sus rutas añadiendo o eliminando puntos, o editar sus marcadores personalizándolos en cualquier momento.

Se podría llevar a cabo esta funcionalidad a través de una interfaz web en la que aparezcan todos los mapas, rutas y mapas de marcadores del usuario y desde la cual pudiera gestionar el contenido.

Otra funcionalidad que de cara al usuario puede ser interesante es:

- **Elección de tipo de rutas**

La aplicación está pensada para devolver siempre la ruta más rápida en coche o bici, pero quizá al usuario le interese para ciertas rutas elegir una ruta a pie aunque no sea la más corta.

Otra posibilidad que ofrece OSM es la opción de obtener direcciones de la ruta desde el punto de salida hasta el último que por defecto se muestra pero de nuevo al usuario quizá no le interese.

- **Opción compartir**

Cada vez es más común que cualquier aplicación incluya la opción de compartir contenido con otras redes sociales. Esta pequeña funcionalidad podría incorporarse a la opción de la gestión web para que el usuario pueda compartir por email, Facebook, Twitter, etc. sus mapas con el resto del mundo.

Capítulo 7

Glosario

AJAX: Asynchronous JavaScript And Xml
API: Application Programming Interface
CU: Casos de uso
GPX: GPS eXchange Format
HTML: HyperText Markup Language
HTTP: Hypertext Transfer Protocol
IDE: Integrated Development Environment
J2EE: Java Platform Enterprise Edition
J2ME: Java Platform Micro Edition
J2SE: Java Platform Standard Edition
JAR: Java Archive
JDK: Java Development Kit
JSON: JavaScript Object Notation
OGC: Open Geospatial Consortium
OSM: OpenStreetMaps
REST: Representational State Transfer
RSS: Really Simple Syndication
SIG: Sistema de información geográfica (GIS en inglés)
URL: Uniform Resource Protocol
WMS: Web Map Service
XML: eXtensible Markup Language

Anexo I:

Referencias

[1] Olaya Ferrero, Víctor.*Sistemas de Información Geográfica (Tomo II)*

[2] Olaya Ferrero, Víctor. *http://volaya.github.io/libro-sig/chapters/Tipos_datos.html*

[3] Olaya Ferrero, Víctor. *http://volaya.github.io/libro-sig/chapters/Introduccion_datos.html*

[4] *https://en.wikipedia.org/wiki/Open_Geospatial_Consortium*

[5] *http://www.mailxmail.com/curso-desarrollo-aplicaciones-dispositivos-inalambricos-j2me/introduccion*

[6] *https://es.wikipedia.org/wiki/OpenStreetMap*

[7] *http://wiki.openstreetmap.org/wiki/ES:Develop*

[8] *https://es.wikipedia.org/wiki/Desarrollo_en_espiral*

[9] *http://normalizacion-equipo2.blogspot.com.es/*

Bosque Sendra, J. (1992). Sistemas de Información Geográfica. Rialp. Madrid.

Lenguajes de programación y herramientas. Información general.

http://es.wikipedia.org

Leaflet.

http://leafletjs.com/reference.html

OpenStreetMaps API. API y todo lo relacionado con desarrollo de OSM

http://wiki.openstreetmap.org/wiki/API_v0.6

http://wiki.openstreetmap.org/wiki/ES:Develop

http://wiki.openstreetmap.org/wiki/Nominatim

http://wiki.openstreetmap.org/wiki/Routing/online_routers

http://wiki.openstreetmap.org/wiki/Downloading_data

Ciclo de Vida Software

http://ciclodevidasoftware.wikispaces.com/

Anexo II:

Apéndices

Manual de instalación IDE (Eclipse)

El IDE utilizado para desarrollar la aplicación es Eclipse, para este proyecto la versión Luna.

Se ha decidido elegir este IDE por el gran número de herramientas para el desarrollo de aplicaciones en Java de las que dispone y la facilidad de uso de las mismas.

A continuación una breve explicación de los pasos seguidos para la instalación del programa:

- Desde la página oficial de Eclipse, navegamos hasta la sección de descargas y elegimos la versión Luna y seleccionamos el sistema operativo para el que se va a realizar la instalación.

https://eclipse.org/downloads/packages/release/luna/sr2

- Para este proyecto se ha trabajado con Windows en 64bits.

Figura 44. Selección de version para instalación Eclipse

- Una vez descargado, se descomprime el contenido del archivo en una carpeta:

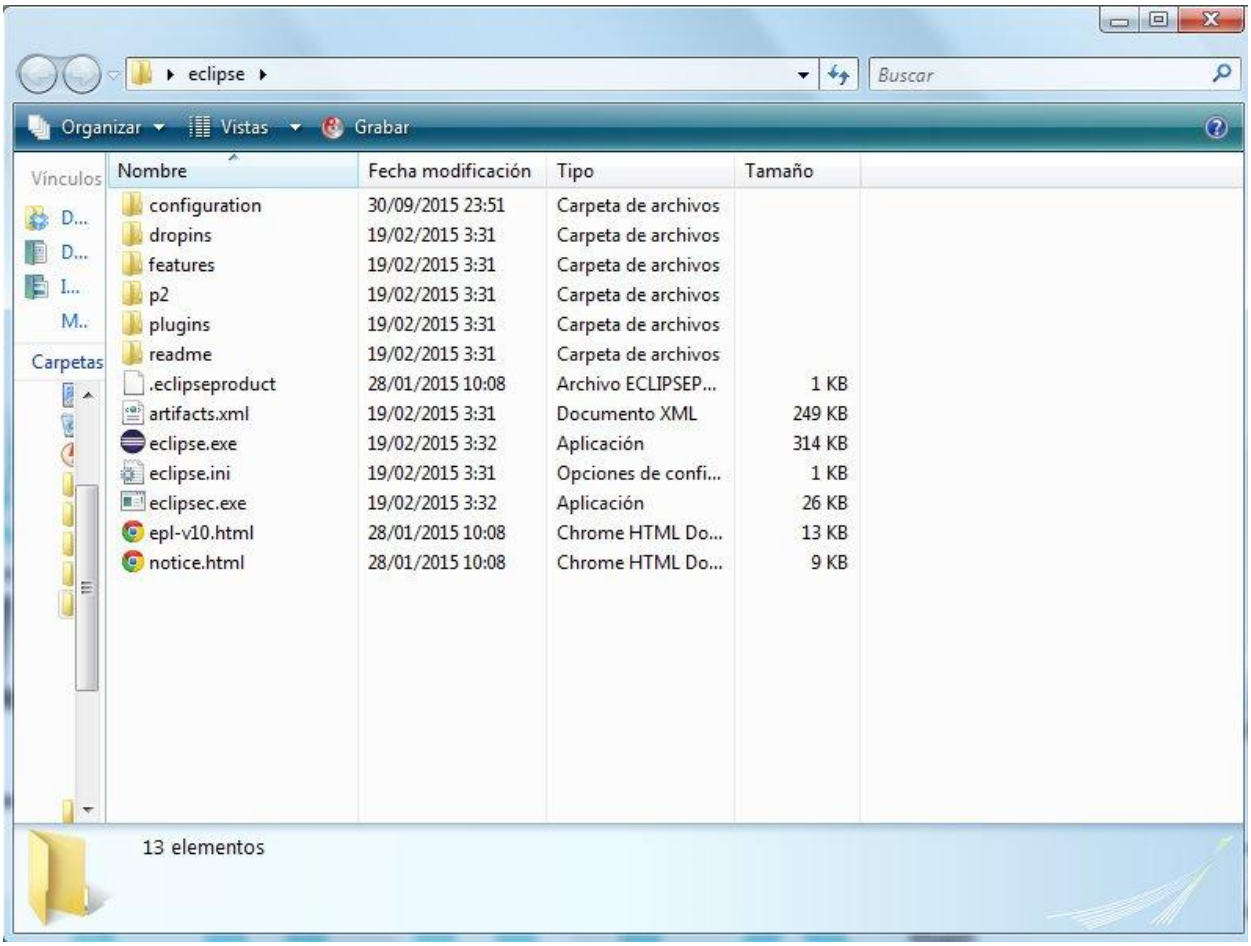


Figura 45. Contenido de la descarga del software descomprimido.

- El siguiente paso tras la ejecución del software es la selección de un directorio que será nuestro espacio de trabajo, donde quedará almacenado todo lo que hagamos en Eclipse.

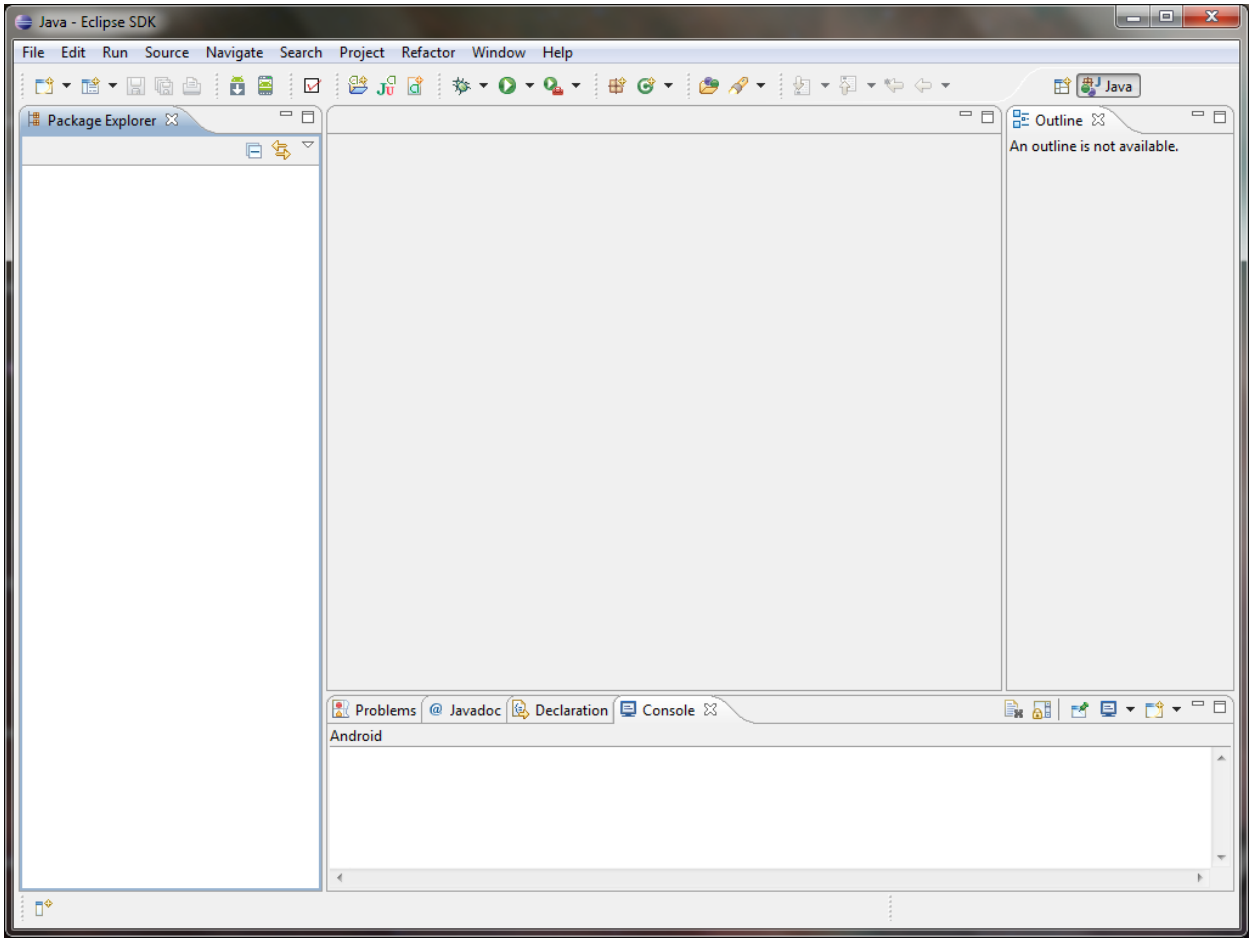


Figura 46. Interfaz IDE Eclipse

Manual de instalación Java y JDK

Es posible que cuando ejecutemos Eclipse por primera vez, aparezca un error si no tenemos instalada la maquina virtual de Java.

Para instalar la maquina virtual de Java entramos la página oficial y seleccionamos la descarga de la versión gratuita:

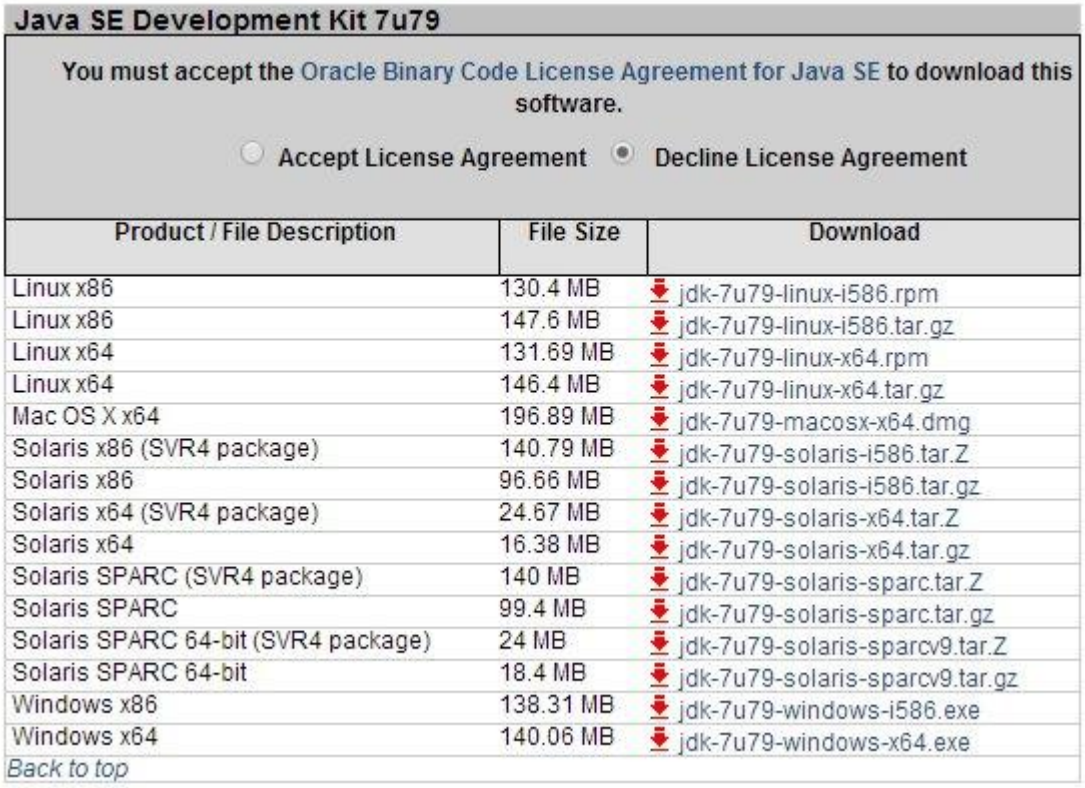
http://java.com/es/download/

Para evitar posibles problemas de compatibilidad con el usuario en caso de que no tenga instalada la última versión de Java en su equipo, se ha elegido la versión 7 de la misma, la cual, en caso de que el usuario si tenga la última versión, también será soportada por la versión 8.

Habrá que instalar, además, el JDK correspondiente a la versión de la maquina virtual que hemos elegido, es decir, la versión 7.

El JDK de Java sirve para desarrollar Software Java y nos permite disponer de herramientas adicionales. JDK es un entorno de desarrollo para hacer aplicaciones, applets, componentes Java, etc…

Desde la página de Oracle, accedemos a sus versiones anteriores a la actual y seleccionamos la que necesitamos:



Java SE Development Kit 7u79

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	130.4 MB	jdk-7u79-linux-i586.rpm
Linux x86	147.6 MB	jdk-7u79-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u79-linux-x64.rpm
Linux x64	146.4 MB	jdk-7u79-linux-x64.tar.gz
Mac OS X x64	196.89 MB	jdk-7u79-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.79 MB	jdk-7u79-solaris-i586.tar.Z
Solaris x86	96.66 MB	jdk-7u79-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.67 MB	jdk-7u79-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u79-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140 MB	jdk-7u79-solaris-sparc.tar.Z
Solaris SPARC	99.4 MB	jdk-7u79-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u79-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.4 MB	jdk-7u79-solaris-sparcv9.tar.gz
Windows x86	138.31 MB	jdk-7u79-windows-i586.exe
Windows x64	140.06 MB	jdk-7u79-windows-x64.exe

Back to top

Figura 47. Tabla de versiones de Java JDK para sistemas operativos.

De nuevo seleccionamos la versión de 64bits para Windows.

Al ejecutar ahora de nuevo Eclipse y a no ser que haya conflicto con otras versiones de Java que se hayan instalado previamente, el IDE ya está configurado para comenzar a trabajar. En caso contrario, si nos aparece un error del tipo:

"A Java Runtime Environment (JRE) or Java Development Kit (JDK) must be available in order to run Eclipse. No Java virtual machine was found after searching the following locations: C:Users…..eclipsejrebinjavaw.exe javaw.exe in your current PATH"

Habrá que configurar las variables de entorno del sistema, a las cuales accedemos desde Mi PC > Propiedades > Opciones avanzadas del Sistema > Variables de entorno.

A continuación, se crea una nueva variable haciendo click en el botón Nueva… en la parte inferior del recuadro de las Variables del Sistema.

Al crear la variable para todo el sistema, afectará a todos los usuarios del equipo. Si solo se quisiera hacer esta configuración para el usuario actual, crearíamos la variable en la sección superior 'Variables de usuario para USER'.

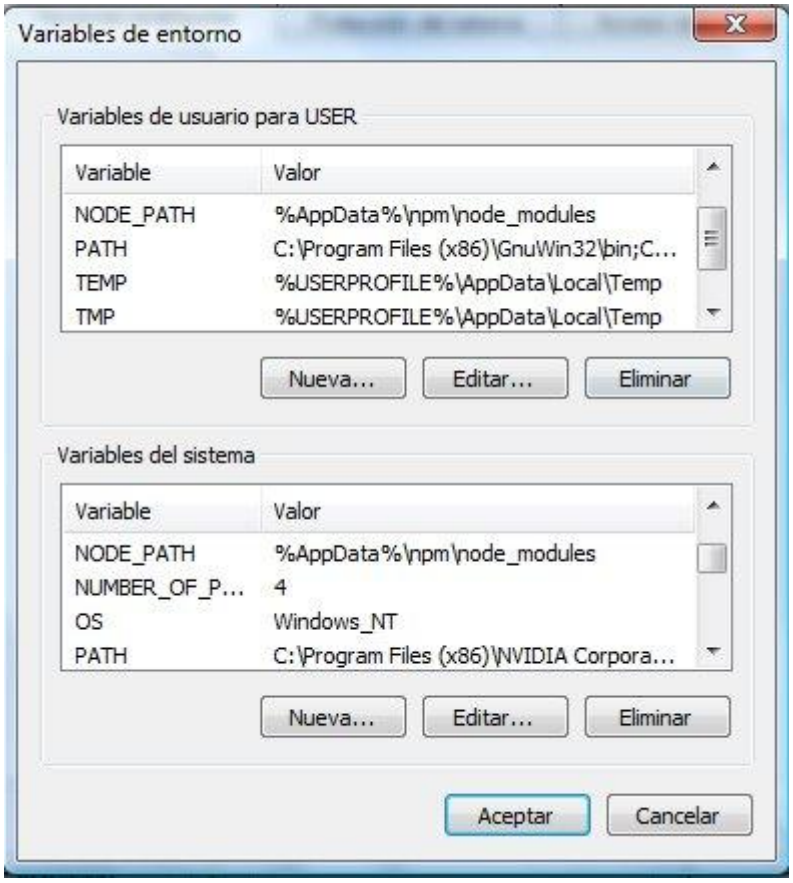


Figura 48. Ventana de variables de entorno.

En el nombre de la variable seleccionamos JAVA_HOME, y en el valor de la variable la ruta en la que se ha instalado el JDK:

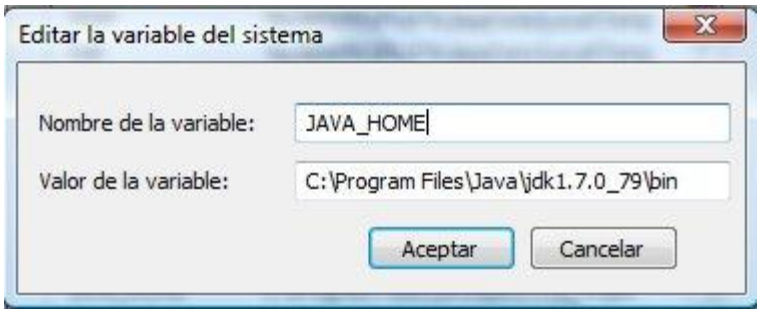


Figura 49. Ventana de creación de la variable del sistema JAVA_HOME.

Manual de usuario

Estructura de la carpeta del proyecto

Para que la aplicación (API.jar) funcione correctamente, debe estar en una carpeta con la siguiente estructura y archivos:

```
API_OSM
|   API.jar
|   marcadores.html
|   template.html
+---icons
|       aeropuerto.png
|       calle.png
|       casa.png
|       cine.png
|       deportes.png
|       fuente.png
|       hospital.png
|       hotel.png
|       iglesia.png
|       monumento.png
|       pAmarillo.png
|       pAzul.png
|       pGris.png
|       pMorado.png
|       pRojo.png
|       pVerde.png
|       restaurante.png
|       ruinas.png
|       universidad.png
+---lib
\---output
        mapa.xml
        output.xml
        ruta.gpx
```

Figura 50. Listado del contenido de la carpeta del proyecto

API.jar : Archivo aplicación

marcadores.html : Archivo de salida que contiene los marcadores que haya introducido el usuario en forma de página HTML.

template.html : Archivo plantilla base a partir del cual se genera marcadores.html

icons : Carpeta que contiene la lista de iconos que el usuario puede usar para personalizar los marcadores.

lib: Carpeta que contiene las librerías Javascript de Leaflet.

Output: Carpeta que contiene los archivos XML y GPX de salida.

mapa.xml: Archivo que contiene el mapa que genera la función location_xml.

output.xml: Archivo que puede guardar tiempos de ruta, los resultados de la función showpoint y en caso de que se produzcan errores el error en concreto.

ruta.gpx: Archivo que contiene la ruta generada con la función routing_gpx

Como ejecutar la aplicación

Para poder ejecutar la aplicación, hay que seguir los siguientes pasos:

1. Abrir una ventana de comandos del sistema.
2. Navegar desde esta ventana hasta el directorio en el que se encuentra la carpeta del proyecto API_OSM.
3. Escribir la siguiente sentencia de comandos:

```
java –jar API.jar API_OSM […]
```

Donde los tres puntos […] corresponderán a la función que se quiera ejecutar seguido de los parámetros necesarios para esa función como se explicara en el punto [9.1.2.1. Ejecución de funciones]

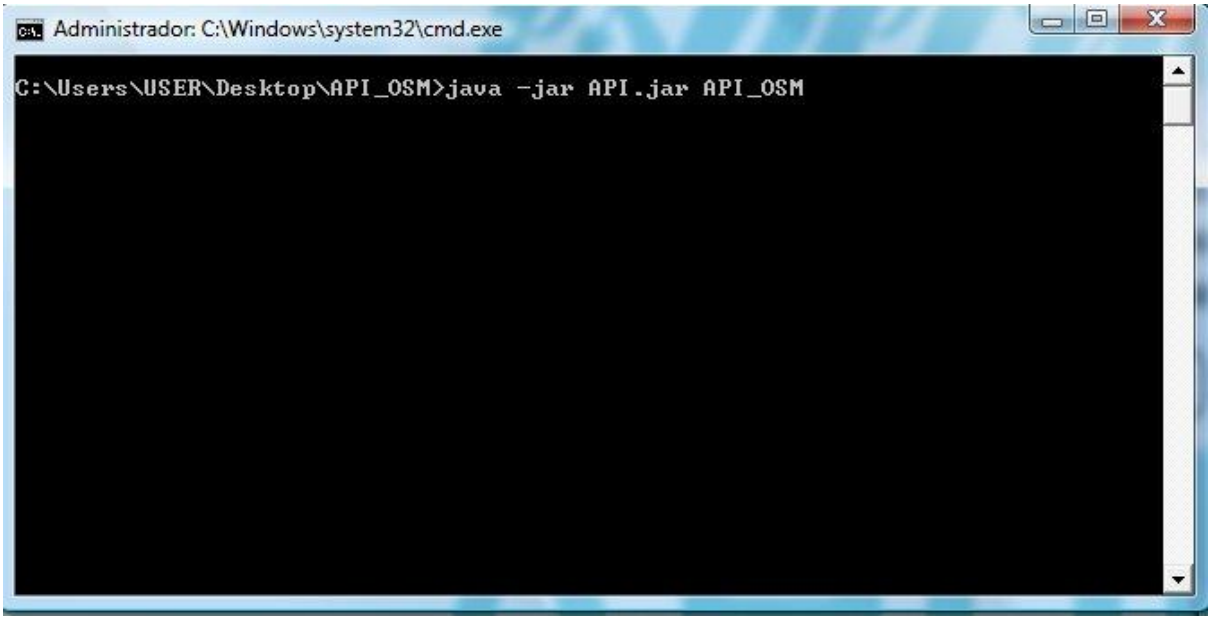


Figura 51. Interfaz consola de comandos segunda solución

Ejecución de funciones

Los parámetros que necesita cada una de las funciones para su correcto uso son:

Función showpoint

java –jar API.jar API_OSM showpoint lugar

java –jar API.jar API_OSM showpoint Av._de_la_Universidad,_Leganés

Función location y location_xml

java –jar API.jar API_OSM (location|location_xml) (lugar|coordenada)

java –jar API.jar API_OSM location Av._de_la_Universidad,_30,_28911_Leganes,_Madrid

java –jar API.jar API_OSM location_xml
Av._de_la_Universidad,_30,_28911_Leganes,_Madrid

Función routin y rountin_gpx

java –jar API.jar API_OSM (routing|routing_xml) (lugar|coordenada)$^{+}$

java –jar API.jar API_OSM routing Paris,_Francia Versalles,_Francia Amsterdam,_Paises_Bajos Berlin,_Alemania Praga,_Republica_Checa Fussen,_Alemania Venecia,_Italia Florencia,_Italia Roma,_Italia

java –jar API.jar API_OSM routing_xml Paris,_Francia Versalles,_Francia Amsterdam,_Paises_Bajos Berlin,_Alemania Praga,_Republica_Checa Fussen,_Alemania Venecia,_Italia Florencia,_Italia Roma,_Italia

Función drawmark

java –jar API.jar API_OSM drawmark ((lugar|coordenada)$^{+}$ [“:” icono] [“:” descripción])$^{+}$

java –jar API-jar API_OSM drawmark Trevi_Fountain,_Italia:fuente:Fontana_de_Trevi Coliseo_Roma,_Italia:monumento:Coliseo Ciudad_del_Vaticano:iglesia:Vaticano Castillo_Sant'Angelo,_Italia:monumento:Castillo_de_Sant_Angelo

Donde:

- Los parámetros separados por el símbolo **“|”** implican que se elige uno u otro.
- Los parámetros dentro de paréntesis **“(“ “)”** se escriben una vez.
- Los parámetros entre corchetes **“[“ “]”** son opcionales.
- Los parámetros entre paréntesis y con el símbolo **“+”** al final se escriben una o más veces.

Reglas de sintaxis

Al ser una aplicación ejecutada desde la consola de comandos del sistema, se debe prestar atención a no usar palabras o comandos reservados por el sistema y que pudieran coincidir con el nombre de algún lugar o descripcion. Por lo cual se decidió que para evitar esto, los parámetros se separan entre sí mediante 1 espacio y los parámetros que contengan varias palabras se sustituyen por una sola palabra sustituyendo los espacios que pudiera contener esta cadena por el símbolo barra baja **“_”**.

Ejemplo:

Parámetro de una sola palabra:

```
parámetro
```

Parámetro con varias palabras:

```
esto_es_un_parámetro_con_varias_palabras
```

Lista de iconos

La lista de iconos que el usuario puede elegir para personalizar los marcadores es la siguiente:

aeropuerto.png
calle.png
casa.png
cine.png
deportes.png
fuente.png
hospital.png
hotel.png
iglesia.png
monumento.png
pAmarillo.png
pAzul.png
pGris.png
pMorado.png
pRojo.png
pVerde.png
restaurante.png
ruinas.png
universidad.png

Archivos de salida

A continuación la lista de archivos de salida que genera cada función:

Función showpoint: Genera el archivo de salida output.xml dentro de la carpeta output. En caso de error se sobrescribirá este archivo con los datos del error.

Función location: No genera archivo de salida salvo error (output.xml)

Función location_xml: Genera el archivo de salida mapa.xml en la carpeta output. En caso de error se genera el archivo con los datos del error.

Función routing: No genera archivo de salida salvo error (output.xml)

Función routing_gpx: Genera el archivo de salida ruta.gpx en la carpeta output. En caso de error se genera el archivo con los datos del error.

Función drawmark: Genera el archivo marcadores.html en la carpeta raíz del proyecto.